(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
29 November 2001 (29.11.2001)

PCT

(10) International Publication Number
WO 01/90883 A2

(51) International Patent Classification[7]: G06F 9/00

(21) International Application Number: PCT/US01/15120

(22) International Filing Date: 9 May 2001 (09.05.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:

60/202,975	9 May 2000 (09.05.2000)	US
60/208,011	26 May 2000 (26.05.2000)	US
60/209,430	2 June 2000 (02.06.2000)	US
60/209,140	2 June 2000 (02.06.2000)	US
60/209,525	5 June 2000 (05.06.2000)	US
09/672,200	27 September 2000 (27.09.2000)	US

(71) Applicant: SUN MICROSYSTEMS, INC. [US/US]; 901 San Antonio Road, Palo Alto, CA 94303 (US).

(72) Inventors: SLAUGHTER, Gregory, L.; 3326 Emerson St., Palo Alto, CA 94306 (US). SAULPAUGH, Thomas, E.; 6938 Bret Harte Dr., San Jose, CA 95120 (US). TRAVERSAT, Bernard, A.; 2055 California St., Apartment 402, San Francisco, CA 94109 (US). ABDELAZIZ, Mohamed, M.; 78 Cabot Ave., Santa Clara, CA 95051 (US).

(74) Agent: KOWERT, Robert, C.; Conley, Rose & Tayon, P.C., P.O. Box 398, Austin, TX 78767-0398 (US).

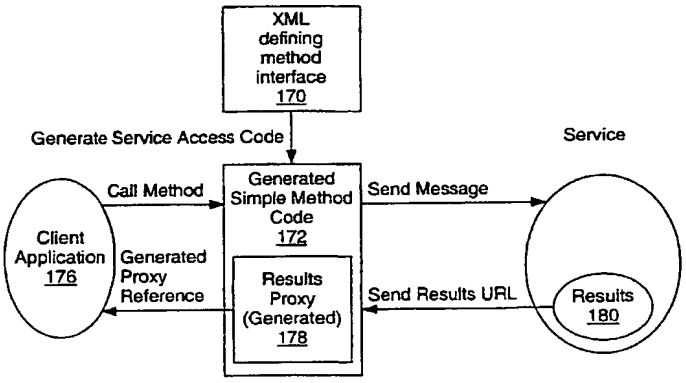
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *without international search report and to be republished upon receipt of that report*

[Continued on next page]

(54) Title: REMOTE FUNCTION INVOCATION WITH MESSAGING IN A DISTRIBUTED COMPUTING ENVIRONMENT



(57) Abstract: An interface between clients and services in a distributed computing environment is described. Method gates may provide an interface to remotely invoke functions of a service. A method gate may be generated from an advertisement that may include definitions for one or more messages for remotely invoking functions of the service. A client may generate messages containing representations of method calls. The service may invoke functions that correspond to the set of messages. A method gate on the service may unmarshal the message and invoke the function. The client may receive the results of the function directly. Alternatively, the results may be stored, an advertisement to the results may be provided, and a gate may be generated to access the results. Message gates may perform the sending and receiving of the messages between the client and service. In one embodiment, functions of the service may be computer programming language (e.g. Java) methods. In one embodiment, a message including a representation of a method call may be generated when no actual method call was made. In one embodiment, a method call may be transformed into messages that may be sent to the service; the service may not know that the messages were generated from a method call. In one embodiment, a service may transform messages requesting functions into method calls; the client may not know that the service is invoking methods to perform the functions.

WO 01/90883 A2

TITLE: REMOTE FUNCTION INVOCATION WITH MESSAGING IN A DISTRIBUTED COMPUTING ENVIRONMENT

BACKGROUND OF THE INVENTION

1. Field of the Invention

This invention relates to distributed computing environments including Web-centric and Internet-centric distributed computing environments, and more particularly to a heterogeneous distributed computing environment including a remote function invocation mechanism based upon a message passing model for connecting network clients and services.

2. Description of the Related Art

Intelligent devices are becoming increasingly common. Such devices range from smart appliances, personal digital assistants (PDAs), cell phones, lap top computers, desktop computers, workstations, mainframes; even, super computers. Networks are also becoming an increasingly common way to interconnect intelligent devices so that they may communicate with one another. However, there may be large differences in the computing power and storage capabilities of various intelligent devices. Devices with more limited capabilities may be referred to as small footprint devices or "thin" devices. Thin devices may not be able to participate in networks interconnecting more capable devices. However, it may still be desirable to interconnect a wide variety of different types of intelligent devices.

The desire to improve networking capabilities is ever increasing. Business networks are expanding to include direct interaction with suppliers and customers. Cellular phones, personal digital assistants and Internet-enabled computers are commonplace in both business and the home. Home networks are available for interconnecting audio/visual equipment such as televisions and stereo equipment to home computers, and other devices to control intelligent systems such as security systems and temperature control thermostats. High bandwidth mediums such as cable and ASDL enable improved services such as Internet access video on demand, e-commerce, etc. Network systems are becoming pervasive. Even without a formal network, it is still desirable for intelligent devices to be able to communicate with each other and share resources.

Currently, traditional networks are complex to set up, expand and manage. For example, adding hardware or software to a network often requires a network administrator to load drivers and configure systems. Making small changes to a network configuration may require that the entire network be brought down for a period. In addition, certain intelligent devices may not support the necessary interfaces to communicate on a given network.

What is needed is a simple way to connect various types of intelligent devices to allow for communication and sharing of resources while avoiding the interoperability and complex configuration problems existing in conventional networks. Various technologies exist for improving the addition of devices to a network. For example, many modern I/O buses, such as the Universal Serial Bus, 1394 and PCI, support plug and play or dynamic discovery protocols to simplify the addition of a new device on the bus. However, these solutions are limited to specific peripheral buses and are not suitable for general networks.

Communication between services in a Jini system is accomplished using the Java Remote Method Invocation (RMI). RMI is a Java programming language enabled extension to traditional remote procedure call mechanisms. RMI allows not only data to be passed from object to object around the Jini network, but full objects including code as well. Jini systems depend upon this ability to move code around the network in a form that is encapsulated as a Java object.

Access to services in a Jini system is lease based. A lease is a grant of guaranteed access over a time. Each lease is negotiated between the user of the service and the provider of the service as part of the service protocol. A service may be requested for some period and access may be granted for some period presumably considering the request period. Leases must be renewed for a service to remain part of the Jini system.

Figure 1 illustrates the basic Jini technology stack. The Jini technology defines a distributed programming model 12 (supported by JavaSpaces, leases, and object templates). Object communication in Jini is based on an RMI layer 14 over a TCP/IP capable networking layer 16.

Jini is a promising technology for simplifying distributed computing. However, for certain types of devices, Jini may not be appropriate. The computing landscape is moving toward a distributed, Web-centric service and content model where the composition of client services and content changes rapidly. The client of the future may be a companion type device that users take with them wherever they go. Such a device may be a combination of a cell phone and a PDA for example. It would be desirable for such devices to be able to communicate and share resources with devices that are more powerful, as well as with thinner or less powerful devices.

In addition, with the advent of the Internet and resulting explosion of devices connected to the net, a distributed programming model designed to leverage this phenomenon is needed. An enabling technology is needed that facilitates clients connecting to services in a reliable and secure fashion. Various clients from thick to thin and services need to be connected over the Internet, corporate Internets, or even within single computers. It is desirable to abstract the distance, latency and implementation from both clients and services.

The key challenge for distributed computing technology is to be scalable from powerful thick clients down to very thin clients such as embedded mobile devices. Current distributed computing technologies, such as Jini, may not be scalable enough for the needs of all types of clients. Some devices, such as small footprint devices or embedded devices, may lack sufficient memory resources and/or lack sufficient networking bandwidth to participate satisfactorily in current distributed computing technologies. The low end of the client spectrum, including embedded mobile devices, often have limited or fixed code execution environments. These devices also may have minimal or no persistent storage capabilities. Most small, embedded mobile devices do not support a Java Virtual Machine. Most code-capable small clients run native code only. In addition, most small devices have little more than flash memory or battery backed RAM as their sole persistent storage media. The size of the storage is often very small and sometimes read-only in nature. Furthermore, the access time of this type of storage media is often an order of magnitude greater than hard disk access time in clients that are more powerful.

Existing connection technologies, such as Jini, may not be as scalable as desired because they are too big. For example, Jini requires that all participants support Java; however, many small clients may not have the resources for a Java Virtual Machine. Furthermore, due to its use of RMI, Jini requires that clients be able to download code and content. Jini may augment the existing client platform by downloading new classes, which

objects to and from remote locations, some means of serialization/deserialization is needed. Such current facilities in the Java Development Kit (JDK) rely upon the reflection API to determine the content of a Java object, and ultimately that code must consult the Virtual Machine. This code is quite large and inefficient.

The fundamental problems with the current method for doing serialization/deserialization include its size, speed, and object traversal model. Code outside the JVM does not know the structure or graph of a Java object and thus must traverse the object graph, pulling it apart, and ultimately must call upon the JVM. Traditional serialization and reflection mechanisms for storing and moving Java objects are just not practical for all types of devices, especially thinner devices. Some of the difficulties with Java reflection and serialization are that an object's graph (an object's transitive closure) reflection is difficult to do outside the JVM. Serialization is too large, requiring a large amount of code. In addition, serialization is a Java specific object interchange format and thus may not be used with non-Java devices.

The Jini distributed computing model requires the movement of Java objects between Java devices. Thus, the serialization mechanism itself is not platform independent since it may not be used by non-Java platforms to send and receive objects. Serialization is a homogenous object format – it only works on Java platforms. Serialization uses the reflection API and may be limited by security concerns, which often must be addressed using native JVM dependent methods. The reflection API may provide a graph of objects, but is inefficient due to the number of calls between the JVM and the code calling the reflection methods.

The use of Java reflection to serialize an object requires an application to ping pong in and out of the JVM to pick apart an object one field at a time as the transitive closure of the object is dynamically analyzed. Deserializing an object using Java deserialization requires the application to work closely with the JVM to reconstitute the object one field at a time as the transitive closure of the object is dynamically analyzed. Thus, Java serialization/deserialization is slow and cumbersome while also requiring large amounts of application and JVM code as well as persistent storage space.

Even for thin clients that do support Java, the Jini RMI may not be practical for thin clients with minimal memory footprints and minimal bandwidth. The serialization associated with the Jini RMI is slow, big, requires the JVM reflection API, and is a Java specific object representation. Java deserialization is also slow, big and requires a serialized-object parser. Even Java based thin clients may not be able to accept huge Java objects (along with needed classes) being returned (necessarily) across the network to the client as required in Jini. A more scalable distributed computing mechanism is needed. It may be desirable for a more scalable distributed computing mechanism to address security concerns and be expandable to allow for the passing of objects, such as Java objects, and even to allow for process migration from one network mode to another.

Object based distributed computing systems need persistent storage. However, as discussed above, attempts at object storage are often language and operating system specific. In addition, these object storage systems are too complicated to be used with many small, embedded systems. For example, the Jini technology uses JavaSpaces as persistent object containers. However, a JavaSpace can only store Java objects and cannot be implemented in small devices. Each object in a JavaSpace is serialized and pays the above-described penalties associated with Java serialization. It may be desirable to have a heterogeneous object repository for distributed computing that may scale from small to large devices.

phones and PDA's with a variety of different networking interfaces, typically low bandwidth. Often these devices have very small displays with limited graphics, but they could include laptops and notebook computers, which may have a larger display and more sophisticated graphics capabilities. The services may be a wide range of applications as well as control programs for devices such as printers. It is desirable for a mobile client to be able to use these services wherever they may be.

A mobile client will often be at a temporary dynamic network address, so networking messages it sends cannot be routed beyond that networking interface (otherwise there may be collisions when two different clients on different networks have the same dynamic address). Mobile clients often do not have the capability for a full function browser or other sophisticated software. The displays may limit the client from running certain applications. Traditional application models are based on predetermined user interface or data characteristics. Any change to the application requires recompilation of the application.

It may be desirable for such clients to have a mechanism for finding and invoking distributed applications or services. The client may need to run even large legacy applications which could not possibly fit in the client's memory footprint. As discussed above, current technology, such as Jini, may not be practical for small footprint devices. The pervasiveness of mobile thin clients may also raise additional needs. For example, it may be desirable to locate services based on the physical location of the user and his mobile client. For example, information about the services in a local vicinity may be very helpful, such as local restaurants, weather, traffic maps and movie information. Similarly, information about computing resources, such as printers in a particular location, may be helpful. Current technologies do not provide an automatic mechanism for locating services based on physical location of the client. Another need raised by thin mobile clients is that of addressing the human factor. Thin mobile clients typically do not contain ergonomic keyboards and monitors. The provision of such human factor services and/or the ability to locate such services in a distributed computing environment may be desirable.

A distributed computing model should provide clients with a way to find transient documents and services. It may be desirable to have a mechanism for finding general-purpose documents (including services and/or service advertisements), where the documents are expressed in a platform-independent and language-independent typing such as that provided by eXtensible Markup Language (XML). Current approaches, including lookup mechanisms for Jini, Universal Plug and Play (UPnP), and the Service Location Protocol (SLP), do not support such a general-purpose document lookup mechanism. For example, the Jini lookup mechanism is limited to Java Language typing and is therefore not language-independent. UPnP and SLP support a discovery protocol only for services, not for general-purpose documents.

SUMMARY OF THE INVENTION

Embodiments of method gates are described that provide a method interface between clients and services in a distributed computing environment. A method gate may be bi-directional, allowing remote method invocations from client to service and from service to client. A method gate may be generated from schema information (e.g. from a service advertisement in a space). The schema information may include definitions for one or more method interfaces. From this information, code may be generated as part of the gate for interfacing to one or more methods. Each method invocation (e.g. from a client application) in the generated code may cause a

above, the results gate may be cast into the Java type that matches the result type. In this embodiment, method gates may be used in the distributed computing environment to allow remote Java objects to behave as local Java objects. The method invocation and results may appear the same to the client Java software program whether the real object is local or remote.

In one embodiment, a message including a representation of a method call may be generated on a client when no actual method call was made. In this embodiment, a process on the client may invoke computer programming language methods on a service without the client actually generating computer programming language method calls.

In one embodiment, a client may transform a method call into one or more messages that may be sent to the service, wherein the messaged do not include information that may be unmarshaled into a method call, but instead include information that may be used by the service to perform a function on behalf of the client. In this embodiment, the service may not know that the messages were originally generated from a method call. These messages may still be considered a "representation" of the method call, but may not be in a form that the service recognizes as a marshaled method call that may be unmarshaled to regenerate a method call.

In one embodiment, a service may transform one or more messages requesting the service to perform one or more functions into a method call. In this embodiment, no method call may have been made on the client to generate the messages. Thus, the client may not know that the service is invoking a method to perform the requested function(s).

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is an illustration of a conventional distributed computing technology stack;

Figure 2 is an illustration of a distributed computing environment programming model according to one embodiment;

Figure 3 is an illustration of messaging and networking layers for a distributed computing environment according to one embodiment;

Figure 4 is an illustration of a discovery service for finding spaces advertising objects or services in a distributed computing environment according to one embodiment;

Figure 5 illustrates client profiles supporting static and formatted messages for a distributed computing environment according to one embodiment;

Figure 6 is an illustration of a distributed computing model employing XML messaging according to one embodiment;

Figure 7 illustrates a platform independent distributing computing environment according to one embodiment;

Figure 8 is an illustration of a distributed computing model in which services are advertised in spaces according to one embodiment;

Figure 9 is an illustration of a distributed computing model in which results are stored in spaces according to one embodiment;

Figure 10a is an illustration of client and service gates as messaging endpoints in a distributed computing model according to one embodiment;

Figure 29 illustrates bridging a client external to the distributed computing environment to a space in the distributed computing environment according to one embodiment;

Figure 30 is an illustration of a proxy mechanism according to one embodiment;

Figure 31 illustrates one embodiment of a client with an associated display and display service according to one embodiment;

Figures 32A and 32B illustrate examples of using schemas of dynamic display objects according to one embodiment;

Figure 33A illustrates a typical string representation in the C programming language;

Figure 33B illustrates an example of a conventional string function;

Figure 33C illustrates an efficient method for representing and managing strings in general, and in small footprint systems such as embedded systems in particular according to one embodiment;

Figure 34 illustrates a process of moving objects between a client and a service according to one embodiment;

Figures 35a and 35b are data flow diagrams illustrating embodiments where a virtual machine (e.g. JVM) includes extensions for compiling objects (e.g. Java Objects) into XML representations of the objects, and for decompiling XML representations of (Java) objects into (Java) objects;

Figure 36 illustrates a client and a service accessing store mechanisms in the distributed computing environment, according to one embodiment;

Figure 37 illustrates process migration using an XML representation of the state of a process, according to one embodiment;

Figure 38 illustrates a mobile client device accessing spaces in a local distributed computing network, according to one embodiment;

Figure 39a illustrates a user of a mobile device discovering the location of docking stations, according to one embodiment;

Figure 39b illustrates a mobile client device connecting to a docking station, according to one embodiment;

Figure 40a illustrates an embodiment of embedded devices controlled by a control system and accessible within the distributed computing environment, according to one embodiment;

Figure 40b illustrates a device control system connected via a network (e.g. the Internet) to embedded devices accessible within the distributed computing environment, according to one embodiment;

Figure 41 is a flow diagram illustrating creating a gate according to one embodiment;

Figure 42a is a flow diagram illustrating a client sending a message to a service according to one embodiment;

Figure 42b is a flow diagram illustrating a service receiving a message from a client and using an authentication service to authenticate the message according to one embodiment;

Figure 42c is a flow diagram illustrating the general process of a client and service exchanging messages with embedded authentication credential according to one embodiment;

Figure 43 is a flow diagram illustrating a mechanism for checking the integrity of messages according to one embodiment;

between sender and receiver. Thus, message transmission may be done asynchronously. In a preferred embodiment, no connection model is imposed. Thus, transports such as TCP are not required. Also, error conditions may be limited to non-delivery or security exceptions.

Messaging layer 104 sits on top of a message capable networking layer 106. In a preferred embodiment, messaging layer 104 does not require that a particular networking protocol be used. TCP/IP and UDP/IP are examples of message capable protocols that may be used for message capable networking layer 106. However, other more specialized protocols such as the Wireless Application Protocol (WAP) may also be used. Other possible message protocols are IrDA and Bluetooth network drivers beneath the transport layer. Networking layer 106 is not limited to a single reliable connection protocol, such as TCP/IP. Therefore, connection to a larger variety of devices is possible.

In one embodiment, message capable network layer 106 may be implemented from the networking classes provided by the Java2 Micro Edition (J2ME) platform. The Java2 Micro Edition platform may be suitable for smaller footprint devices that do not have the resources for a full Java platform or in which it would not be efficient to run a full Java platform. Since J2ME already provides a message capable family of networking protocols (to support sockets), it follows that for the small footprint cost of adding messaging layer 104, distributing computing facilities may be provided for small devices that already include J2ME.

Message capable networking layer 106 may also be provided by the Java Development Kit's (JDK) java.net networking classes. Alternatively, any message capable networking facilities may be used for message capable networking layer 106. In a preferred embodiment, a reliable transport is not required, thus embedded devices supporting an unreliable data gram transport such as UDP/IP may still support the messaging layer.

Thus, thin clients may participate in a distributed computing environment by simply adding a thin messaging layer 104 above a basic networking protocol stack. As shown in Figure 3, a basic system includes messaging layer 104 on top of a networking layer 106. The networking layer may provide for reliable messages, e.g. TCP, or unreliable messages, e.g. UDP. The Internet Protocol (IP) is shown in Figure 3 as an example of protocol that may be used in networking layer 106. However, the distributed computing environment does not require IP. Other protocols may be used in the distributed computing environment besides IP. A network driver such as for Ethernet, Token Ring, Bluetooth, etc. may also be part of the networking layer. Many small clients already provide a network driver and transport protocol such as UDP/IP. Thus, with the addition of the thin XML based messaging layer, the device may participate in the distributed computing environment.

Thus, the foundation for the distributed computing environment is a simple message passing layer implemented on top of reliable connection and/or unreliable data grams. The messaging technology is very different from communications technologies employed in other distribution computing systems, such as Jini which employs the Java remote method invocation (RMI). The message passing layer 104 supports an asynchronous, stateless style of distributed programming, instead of the synchronous, state-full style predicated by RMI. Moreover, message passing layer 104 is based on a data representation language such as XML and thus copies data, but not code, from source to destination, unlike RMI. By using a representation data language, such as XML, messaging layer 104 may interoperate with non-Java and non-Jini platforms in a seamless fashion because Java code is not assumed on the sending or receiving end of a message. Moreover, unlike RMI, messaging layer 104 does not require a reliable transport mechanism such as TCP/IP.

device may use these messages to announce its presence and its status. Such small devices may also receive raw messages to implement certain functions, such as turning a feature on or off.

Message-based services such as spaces may send and receive reliable formatted messages. A space message may be formatted with a well-defined header and with XML. In one embodiment, a formatted message send may occur when a client uses a space method to claim, write, or take objects from a space. The message contents may be dynamically formatted in XML and contain well-defined headers. Figure 5 illustrates client profiles supporting formatted and static messages. By using static messages, small devices may use a smaller profile of code to participate in the distributed computing environment. For example, a small device could just send basic pre-defined messages. Depending on the client, the static pre-defined messages may consume a small amount of memory (e.g. <200 bytes). Static messages may also be an option even for larger devices. On the other hand, the dynamic XML messages may be useful when object values are not known at compile time.

Turning now to Figure 6, a distributed computing model is illustrated that combines a messaging system with XML messages and XML object representation. The platform independence of XML may be leveraged so that the system may provide for a heterogeneous distributed computing environment. Thus, client 110 may be implemented on almost any platform instead of a particular platform like Java. The messaging system may be implemented on any network capable messaging layer, such as Internet protocols (e.g. TCP/IP or UDP/IP). Thus, the computing environment may be distributed over the Internet. In one embodiment, the messaging system may also use shared memory as a quick interprocess message passing mechanism when the client and/or space server and/or service are on the same computer system. The distributed computing model of Figure 6 may also be very scalable because almost any size client can be configured to send and/or receive XML messages.

As shown in Figure 6, two kinds of software programs may run in the distributed computing model: services 112 and clients 110. Services 112 may advertise their capabilities to clients wishing to use the service. The services 112 may advertise their capabilities in spaces 114. As illustrated in Figure 7, clients 110 and services 112 may or may not reside within the same network device. For example, devices 120 and 124 each support one client, whereas service 112a and client 110b are implemented in the same device 122. Also, as illustrated in Figure 7, no particular platform is required for the devices to support the clients and services. For example, device 120 is Java based, whereas device 124 provides a native code runtime environment.

A device may be a networking transport addressable unit. Example devices include, but by no means are limited to: PDAs, cellular/mobile phones, notebook computers, laptops, desktop computers, more powerful computer systems, even supercomputers. Both clients and services may be URI-addressable instances of software (or firmware) that run on devices. Using the distributed computing environment architecture, a client may run a service. A space is a service that manages a repository of XML documents. Even though it is redundant, the term, space service, may be used herein for readability. A software component may be both a client and service at different times. For example, when a service uses a space (e.g. to advertise itself), that service is a client of the space.

Figure 8 illustrates the basic model of the distributed computing environment in one embodiment. The distributed computing environment may connect clients 110 to services 112 throughout a network. The network may be a wide area network such as the Internet. The network may also be a combination of networks such as a local area network (LAN) connected to a wireless network over the Internet. As shown in Figure 8, a service 112

services that enable clients and services to store, advertise, and address content. Clients and services may find each other and broker content using a transient storage space. Services may place a content or service advertisement in a space. The advertisement may describe the content type or the capabilities of the service. Clients may subsequently browse spaces looking for advertisements that match a desired set of capabilities. When a client finds a matching advertisement, a communication channel may be established which may enable bi-directional message passing to the service backing the advertisement. In one embodiment, the communication channel is authenticated. Results (which are just another content type) from service operations may be returned directly to the client in a response message, advertised and stored in a space, or advertised in a space, but stored persistently. Stored results may be addressed using a URI (e.g. returned in the response message) and may have an associated authentication credential.

Message Gates

As discussed above, the distributed computing environment leverages off the use of a data description language, such as XML. XML may be used to describe a target entity (e.g. document, service, or client) to an extent such that code may be generated to access that entity. The generated code for accessing the target entity may be referred to as a message gate. Thus, in one embodiment, the distributed computing environment differs from other distributed computing environments in that instead of passing the necessary code between objects necessary to access the other object, the environment provides access to XML descriptions of an object or target so that code may be generated based on the XML description to access the target. The distributed computing environment may use an XML schema to ensure type safety as well as a programming model (e.g. supported messages) without having to agree upon language specific APIs, just XML schemas.

Code generated from an XML schema may also incorporate the language, security, type safety, and execution environment characteristics of the local platform. The local platform may thus have control over the generated code to ensure that it is bug-free and produces only valid data according to the schema. The generated code may conform to the client's code execution environment (e.g. Java, C++, Smalltalk), as well as its management and security framework (Web-server and/or operating system).

Note that the distributed computing environment does not require that code generated from an XML schema be generated "on the fly" at runtime. Instead, some or all of the code may be pre-generated for categories (or classes) of services, and then linked-in during the platform build process. Pre-generation of code may be useful for some clients, such as embedded devices, where certain XML schemas are already known. In one embodiment, some or all of the code doesn't actually have to be generated at all. A private code-loading scheme (within the client) might be used in one embodiment to augment the generation process. In addition, the distributed computing environment may specify, in some embodiments, an interface to download code for additional features in accessing a service (see, e.g., message conductors described below). Typically, such downloaded code may be small and the client may have the option to download the code or not.

The phrase "generated code" may refer to code that originates within the client under the control of the client code execution environment, or to code that is generated elsewhere (such as on the service system or on a space service system) and that may be downloaded to the client system after generation. Binding time, however,

A message gate may provide a secure communication endpoint that type checks XML messages. As further discussed below, a message gate may also provide a mechanism to restrict the message flow between clients and services. In one embodiment when a client desires to access a service, a client and service message gate pair is created, if not already existing. In one embodiment, the service message gate may be created when the service receives a first message from the client message gate. In one embodiment, one or more service message gates may be created when the service is initialized, and may be used to pair with client message gates when created. The creation of a message gate may involve an authentication service that may negotiate the desired level of security and the set of messages that may be passed between client and service. In one embodiment, the authentication service may accept a client ID token (also referred to as a client token), a service ID token (also referred to as a service token), and a data representation language message schema that describes the set of data representation language messages that may be sent to or received from the service. For example, messages may be described that may be sent from a client to a service to invoke the service or to invoke aspects of the service. Messages may also be described that are to be sent from the service, such as response messages and event notification messages. Refer to the Authentication and Security section below for a further discussion of how the authentication service may be used in the construction and use of message gates.

A client message gate and a service message gate pair may allow messages to be sent between the client and the service. In one embodiment, message gates may be created that only send and/or receive a subset of the total set of messages as described in the message schema for a service. This limited access may be used within the distributed computing environment to implement a policy of least privilege whereby clients are only given access to specific individual message types, based on a security policy. Refer to the Authentication and Security section below for a further discussion of security checks for gate usage and gate creation.

Client and service gates may perform the actual sending (and receiving) of the messages from the client to the service, using the protocol specified in the service advertisement (URI of service in the service advertisement). The client may run the service via this message passing. A message gate may provide a level of abstraction between a client and a service. A client may access a service object through a message gate instead of accessing the service object directly. Since the gate abstracts the service from the client, the service's code may not need to be loaded, and then started, until the client first uses the service.

The client gate may also perform verification of the message against the XML schema, or verification of the message against the XML schema may be performed by the service gate, e.g. if the client indicates it has not yet been verified. In some embodiments, verification may not be practical for simple clients and may thus not be required at the client. In some embodiments, verification may be performed by the service. The gates may also perform authentication enablement and/or security schemes. In one embodiment, if a client does not support the protocol specified in the service advertisement, then it may not be able to construct the right gate. To avoid this problem, service advertisements (used for gate construction) may include a list of possible URIs for a service, so a variety of clients may be supported.

A basic message gate may implement an API to send and receive messages. The API moves data (e.g. XML messages) in and out of the gate, validating messages before sending and/ or upon receiving. In one embodiment, message gates may support a fixed minimum API to send and receive messages. This API may be extended to other features as discussed below. As illustrated in Figure 10b, a gate 130 may be generated

A gate factory for a device may generate gate code that may incorporate the language, security, type safety, and/or execution environment characteristics of the local device platform. By constructing gates itself, a device has the ability to ensure that the generated gate code is bug-free, produces only valid data, and provides type-safety. An advantage of a device generating its own gate code as opposed to downloading code for accessing a service is that the client code management environment has the control. The generated code may conform to the client's code execution environment (e.g. Java, C++, Smalltalk), as well as its management and security framework (Web-server and/or operating system). Generated code is also trusted code, because the client's runtime environment was involved in its creation. Trusted security information therefore may also be added by the trusted generated code. Thus, a device may receive an XML message schema for a service and then construct a gate based on that schema to access the device. The XML schema may be viewed as defining the contract with the service and the generated gate code as providing a secure way to execute the contract. Note that open devices, in which un-trusted (e.g. downloaded) code may be run, may be configured so that gates may be generated only by trusted code. Open devices may employ a process model in which gates are enclosed in a protected, isolated code container that is not accessible to tools, such as debuggers, capable of discovering the gate's implementation, especially the gates authentication credential.

A gate factory 140 may negotiate on behalf of a client with a service to create a gate to send messages to the service. Similarly, a gate may be constructed at the service to receive messages from the client gate and send messages to the client gate. Together, the client and service gates may form a secure bi-directional communication channel.

A gate factory may provide a level of abstraction in gate creation. For example, when a client desires to use a service, instead of the client directly creating a gate to access the service, the gate may be created by a gate factory as part of instantiating the service.

The gate factory may create or may include its own trusted message gate that is used to communicate with an authentication service (e.g. specified by a service advertisement) to receive an authentication credential for the gate being constructed. For services that do not restrict access, a gate may be constructed without an authentication credential. The gates for such services may not need to send an authentication credential with each message since the service does not restrict access. The authentication service is an example of a service that does not restrict access, in one embodiment. Thus, a gate factory may be configured to optimize gate construction by checking whether a service restricts access. If the service does not restrict access, then the gate factory may avoid running an authentication service as part of gate construction and may avoid included provisions for an authentication credential as part of the constructed gate. The gate factory may also receive or download an XML message schema (e.g. specified by a service advertisement) to create a gate matching that schema. The gate factory may also receive or download a URI for the service and/or for a service message gate for use in creating the client message gate to communicate with the URI.

In addition, another gate construction optimization may be employed for certain clients that do not desire to perform checking of messages against a service's XML schema. The client may be too thin to perform the checking or may rely on the service gate to perform the checking or may simply choose not to perform the checking (e.g. to reduce gate memory footprint). The gate factory may be configured to receive an indication of whether or not a gate should be constructed to verify messages against the provided XML schema. In some

As an example, if it is desired for a device to have a built-in message gate that can send and receive messages from a digital camera, the build of the device software may include running the gate generation tool with the camera's XML message schema as input. The XML schema may be parsed by the XML parser that may convert the XML schema into an internal form suitable for quick access during a message verification process. The tool's code generator may provide source code for a gate corresponding to the camera's schema. In some embodiments, the generation tool may also compile the source code and the gate code may be linked into the software package for the device. At runtime, the camera service may be discovered in the distributed computing environment. The message URI for the camera service may be bound to the built-in gate for the camera within the device. The binding of the URI to the pre-constructed gate may be performed by a gate constructor within the device. This gate constructor may be a much smaller, simpler gate factory. When the camera service is instantiated, the URI for the camera service is passed to the gate constructor as an XML message. The gate constructor may then bind the URI to the pre-constructed gate.

Thus, a gate may be partially or fully generated at runtime, or a gate may be pre-generated before runtime with a binding process (e.g. for a URI or credential) performed at runtime. In one embodiment, a gate generation tool such as the gate factory or the generation tool for pre-constructed gates may be a Java-based tool to provide some level of platform independence. Alternatively, gate generation tools may be provided in any language, such as the native code for a particular device in the distributed computing environment.

Note that the distributed computing environment does not preclude a device from downloading part or all of a gate's code. For example, in some embodiments, a service may provide gate code that may be downloaded by a client wishing to access that service. However, downloaded code may present size, security and/or safety risks.

A more detailed illustration of possible gate components for one embodiment is shown in Figure 12. A gate may include its address (or name) 150, a destination gate address 152, a valid XML schema (or internal form thereof) 154, and a transport URI 153. In other embodiments, a gate may also include an authentication credential 156. Some gates may also include a lease 158 and/or a message conductor 160 to verify message ordering.

A gate's name 150 may be a unique ID that will (for the life of the gate) refer only to it. A gate may be addressed using its gate name 150. In one embodiment, gate names may be generated as a combination of a string from an XML schema (e.g. from a service advertisement) and a random number, such as a 128-bit random number. The name 150 may allow clients and services to migrate about the network and still work together. In a preferred embodiment, the gate address is independent of the physical message transport address and/or socket layer. Thus, a gate name may provide a virtual message endpoint address that may be bound and un-bound to a message transport address. In one embodiment, a gate's name may be a Universal Unique Identifier (UUID) that may, for the life of the gate, refer only to it.

A gate name may persist as long as the gate persists so that different applications and clients executing within the same device may locate and use a particular gate repeatedly. For example, a gate may be created for a first client process executing within a device to access a service. After the first client process has completed its activity with the service, it may release the gate. Releasing the gate may involve un-binding the gate from the first client process's message transport address (e.g. IP and/or Port address). The gate may be stored in a gate cache or repository. A second client process executing within the same device that desires to run the same service may locate the gate by its name and use it to access the service. To use the gate, the second client process may bind the

either the client service messages or the provider service messages. In addition, once a gate has been constructed, a client may query as to the capabilities of the service without the gate actually sending a message, but instead by inspecting the gate's set of messages.

As described above, a message gate may verify the sender of the message using an authentication credential, message content for type safety and according to an XML schema. However, it may also be desirable to verify that messages are sent between a client and a service in the correct order. It may be desirable to be able to provision applications (services) for clients to run without any pre-existing specific functionality related to the application on the client (e.g. no GUI for the application on the client). For example, a Web browser may be used on a client as the GUI for a service instead of requiring an application-specific GUI. Of the possible messages in the XML schema, the client may need to know what message next to send to the service. It may be desirable for the client to be able to determine which message to send next without requiring the client to have specific knowledge of the service. In one embodiment, the service may continually send response messages indicating the next input it needs. The service would then accept only the corresponding messages from the client with the requested input specified. Other ad hoc scheme for message ordering may also be employed.

In another embodiment, a message conductor 160 may be employed in the gate or associated with the gate to verify the correct sequence of messages, as opposed to verifying each message's syntax (which may already be performed in the gate according to the schema). Message conductor 160 may provide a more general approach for application provisioning. The message conductor 160 may be specified in a service's advertisement. The message conductor indication in a schema may allow code to be generated on or downloaded to the client during gate construction, which may provide the choreography needed to decide which message to send next to the service. A message conductor may be implemented as a Java application, a Java Script, WML script, or in other programming or scripting languages.

In one embodiment, the message conductor may accept as input an XML document (e.g. from a service advertisement) that presents the valid order or choreography for messages that may be sent between a client and the service. This XML document may also specify user interface information and other rules. The conductor may parse this XML document into an internal form and enforce message ordering (and/or other rules) according to the enclosed ordering information. The conductor may prevent messages from being sent out of order. Or, if a message is sent out of order, an exception may be raised within the sending device. If a message is received out of order, the conductor may send an automatic response message back declaring the ordering error. The sender may then resend messages in the correct order. Note that in some embodiments, part or all of a conductor may be shared by several gates. Thus, a conductor may be linked to multiple gates.

In one embodiment of a distributed computing environment, front ends for services (service interfaces) may be built in to clients. In one embodiment, the service interface may be a preconstructed user interface provided to the client by the service. In one embodiment, the service interface may be provided to the client in the service advertisement. The service interface may interact on the client with the user of the service to obtain input for running the service, and then may display results of running the service on the client. A "user" may be a human, embedded system, another client or service, etc. In one embodiment, a client device may not be able to provision arbitrary services, as the client device may only be able to run services for which it has a front end built in. In one embodiment, a service interface for a service may be implemented in a Web browser on the client.

example. If a received message includes an OFF tag, the receiving gate will stop sending messages to its paired destination gate. If the gate receives a message including an ON tag, it may resume sending messages.

Thus, a service-side gate may monitor the use of its resources and trigger flow control if use of its resources exceeds a threshold. For example, a service may reduce its load by sending messages including OFF tags to one or more client gates. The client gates receiving the messages with OFF tags will stop sending messages to the service. Pending messages in the clients may be buffered or may be handled by internal flow control mechanisms. Once the service is able to handle more requests, it may send messages to one or more clients with ON tags so that the clients may resume sending messages. In other embodiments, other flow control tags may be supported in addition to or instead of ON and OFF. Other flow control tags may indicate to reduce message flow or that message flow may be increased.

Message gates may be configured to perform resource monitoring. For example, since all messages may flow through a gate, the gate may be configured to manage and/or track a client's use of a service (and possibly its associated resources such as memory or threads). A gate may be configured to track the activity of a software program, such as a client, by monitoring how much a resource, such as a service, is used or which and how many service resources are used. In one embodiment, a gate may generate or may facilitate generation of a client activity log. Each message and its destination or sender may be logged.

A gate may also be configured to perform resource monitoring for flow control from the local (sending) side of a gate pair. If the client exceeds an allocated bandwidth of service (or resource) usage, the gate may automatically throttle back the flow of messages, for example. Thus, a client-side message gate may automatically trigger different flow control modes by monitoring the flow of outgoing messages. If the outgoing message flow exceeds a threshold, the gate may reduce or shut off its flow of outgoing messages. The threshold may be specified in a service's XML schema or advertisement. In some embodiments, the threshold may be specified only for messages using certain service resources or for all messages.

The gate may also be configured to determine when message flow may be increased or resumed. In one embodiment, the gate may maintain a count of outgoing messages that have been sent without the matching reply (response) received. When matching responses are received by the client-side gate, the count of outstanding request messages may be decremented. When the counts decrements below a specified outstanding request message threshold, the gate may increase or resume sending new request messages.

A gate may be configured to support message-based accounting and/or billing. A billing system may be implemented based upon the number and/or kind of messages sent and/or received by a message gate. Since all messages to and from a client may pass through a gate, the gate may be configured to facilitate charging a client for service usage, for example on a per message basis or "pay as you go". Thus, a billing system may be implemented within the distributed computing environment in which a user could be charged, for example, each time a message is sent and/or received by software running on behalf of the user.

In one embodiment, a message gate may receive billing information from an XML schema, e.g. for a service. The billing information may denote a billing policy and a charge-back URI. The charge-back URI may be used by the message gate to charge time or usage on behalf of a user. A message gate may make a charge-back by sending a charge message to the charge-back URI specified in the XML schema. Gates so configured may be referred to as bill gates. The billing policy may indicate charge amounts per message or per cumulative message

172 may be generated from XML schema information 170 (e.g. from a service advertisement in a space). The XML schema information 170 includes XML defining a method interface(s). From this information, code may be generated as part of the gate for interfacing to one or more methods. Each method invocation (e.g. from a client application 176) in the generated code may cause a message to be sent to the service containing the marshaled method parameters. The message syntax and parameters to be included may be specified in the XML schema. Thus, the method gate 172 provides an XML message interface to remotely invoke a service method. The method gate may be generated on the client or proxied on a server, such as the space server where the service method was advertised or a special gateway server.

A service may have a corresponding method gate that implements or is linked to a set of object methods that correspond to the set of method messages defined in the service's XML schema. There may be a one to one correspondence between the object methods implemented by or linked to the service's method gate and the method messages defined by the service's XML schema. Once a service's corresponding method receives a message from a client to invoke one of the service's methods, the service's method gate may unmarshal or unpack the parameters of the message invocation and then invoke the method indicated by the received message and pass the unmarshalled parameters.

The method gate may provide a synchronous request-response message interface in which clients remotely call methods causing services to return results. The underlying message passing mechanics may be completely hidden from the client. This form of remote method invocation may deal with method results as follows. Instead of downloading result objects (and associated classes) into the client, only a result reference or references are returned in XML messages, in one embodiment. An object reference 178 may be a generated code proxy (e.g. results gate) representing the real object result 180 (still stored out on the net, for example). In other embodiments, the client may choose to receive the actual result object. In addition, once a client has received a result object reference, the client may use this reference to receive or manipulate the actual result object. In one embodiment, the result reference includes one or more URIs to the real result.

The real result object(s) may be stored in a service results space (which may be created dynamically by a servlet, for example). This temporary results space may act as a query results cache. The results cache (space) may be patrolled by server software (garbage collector) that cleans up old result areas. Results returned from each method invocation may be advertised in the results space. A result itself may be or may include a method that could then be remotely instantiated by a client, thus generating its own method gate. Therefore, the distributed computing environment may support recursive remote method invocation.

As mentioned above, when a client uses a method gate to remotely invoke a service method, a reference to the method results may be returned from the service method gate instead of the actual results. From this reference, a results gate may be generated to access the actual result. Thus, the client or client method gate may receive a result URI and perhaps a result XML schema and/or authentication credential for constructing a gate to access the remote method results.

In one embodiment, a service gate may create a "child gate" for the results. This child results gate may share the same authentication credential as its parent gate. In some embodiments, results may have a different set of access rights and thus may not share the same authentication credential as its parent. For example, a payroll service may allow a different set of users to initiate than to read the payroll service's results (paychecks).

on the client device. In one embodiment, the message may be in a data representation language. In one embodiment, the data representation language is XML. In one embodiment, a client process may make a method call, and the method call may be marshaled into a representation of the method call included the message. The representation of the method call in the message may include, but is not limited to, an identifier (e.g. method name) that identifies the method call, and zero or more parameter values for the method call. The representation of the method may be in the data representation language. For example, the elements in the representation may be one or more tags (e.g. XML tags) in the message.

In one embodiment, a method gate may access the method call generated by the client process, and then may marshal the method call into the message. In one embodiment, a message including a representation of a method call may be generated on the client when no actual method call was made. In this embodiment, a process on the client may invoke computer programming language methods on a service without actually generating computer programming language method calls. For example, a client may include one or more "canned" messages that the client may send to the service to invoke one or more methods of the service. Thus, the client may use messages to request the service to perform methods without a process on the client actually making a method call.

At 2102, the client may send the message to the service. In one embodiment, the client method gate may send the message to the service. In one embodiment, the client method gate may send the message directly to the service. In another embodiment, the client method gate may provide the message to a client message gate, which may then send the message to the service.

At 2104, after the service receives the message, a function may be performed on the service in accordance with the representation of the method call in the received message. In one embodiment, the service may unmarshal or unpack the method call from the received message, invoke the method indicated by the received message and pass the unmarshaled parameter values to the invoked method. In one embodiment, the unmarshaling and invoking may be performed by a method gate on the service. In one embodiment, instead of invoking the method, the service may perform a function (e.g. other method or some other action) in response to the message. For example, the service may execute a function that generates the desired results of the method call without invoking a method on the service. In another example, the service may have previously generated results data for the method call, and may provide the results data to the client in response to the message without invoking the method.

At 2106, the function on the service may generate results data in response to the message. At 2108, the results data may be provided to the client. In one embodiment, the results data may be returned directly to the client (e.g. in a message). In another embodiment, the results data may be stored in the distributed computing environment (e.g. on the service device, or on a space service external to the service device) and an advertisement may be provided to the client for accessing the stored results data. The client may then use the advertisement to set up a results gate for accessing the results data. In another embodiment, the service may return a results gate to the client for accessing the results data.

In one embodiment, the client may transform a method call into one or more messages that may be sent to the service, wherein the messaged do not include information that may be unmarshaled into a method call, but instead include information that may be used by the service to perform a function on behalf of the client. In this embodiment, the service may not know that the messages were originally generated from a method call. In one

Returning to Figure 45b, at 2116, the service method gate may unmarshal or unpack the method call from the received message, invoke the method indicated by the received message and pass the unmarshaled parameter values to the invoked method. The service method gate may use the identifier included in the message to locate the template for the method call. The service method gate may implement or may be linked to a set of methods that correspond to the set of method messages defined in the service's XML schema. There may be a one to one correspondence between the object methods implemented by or linked to the service's method gate and the method messages defined by the service's XML schema. In one embodiment, a method gate may be created on a client in accordance with a message schema provided to the client by the service, e.g. in a service advertisement. In one embodiment, a gate factory may be used to generate the method gate on the client.

At 2118, the invoked method may be implemented or performed on the service. At 2120, the invoked method may produce results output from the invocation in accordance with the parameter values received in the method invocation. At 2122, the results of the method invocation may be provided to the client process that originally invoked the method. In one embodiment, the service may directly provide the results to the client in one or more results messages. In another embodiment, the service may place the results in a space and may provide a results advertisement to the client for accessing the results. Figure 45d and 45e illustrate other embodiments where a results gate may be provided to the client for accessing the results.

In Figure 45d, at 2130, the service method gate may provide a results reference to the service message gate. In one embodiment, the results reference may be an advertisement to the results. In another embodiment, the results reference may be an address, e.g. a URI, for the results. At 2132, the service message gate may send the results reference to the client message gate in a message. At 2134, the client, after receiving the message, may create a results gate from the results reference message. For example, a results advertisement may be provided in the message, and the client may create the results gate in accordance with the results advertisement in a similar manner to creating message gates from advertisements as described elsewhere herein. At 2136, the client may use the results gate to access the results. For example, the results may be on the service device, and the service may have generated a service results gate. The client results gate may be generated to communicate with the service results gate, e.g. the URI of the service results gate may have been included in the results advertisement. Alternatively, the service may have stored the results elsewhere in the distributed computing environment and the client results gate may be configured to communicate with a results gate set up for the results. For example, the results may be on a space, and the client results gate may communicate with a results gate on the space.

In Figure 45e, the service may create a results gate at 2140. The service may then send the results gate to the client (in a message) at 2142. The received results gate may then be used by the client to access the results. This embodiment frees the client from the responsibility of setting up the results gate.

To the client process that originally invoked the method, the process of RMI as described herein is transparent. To the client process, it appears as if the method is being invoked locally, and that the results are being provided locally. Thus, the method gates, in combination with the message passing mechanism, may allow "thin" clients to execute processes that would otherwise be too large and/or complex to execute on the clients.

A Java platform method gate implementation example follows. The actual Java method gate implementation for any Java platform is defined in the distributed computing environment platform binding. Each Java method gate granting access to a result is actually a Java class instance (an object) that implements the

In one embodiment the event gate automatically subscribes itself for events on behalf of the local consumer clients. As clients register interest with the gate, the gate registers interest with the event producer service. A client may also un-subscribe interest, which causes the gate to un-register itself with the service producing the event.

An event gate may type check the event document using the XML schema just like a regular message gate does in the standard request-response message passing style described above. An event gate may also include an authentication credential in messages it sends and verify the authentication credentials of received event messages.

Note that any combination of the gate functionality described above may be supported in a single gate. Each type has been described separately only for clarity. For example, a gate may be a message gate, a method gate and an event gate, and may support flow control and resource monitoring

Service Discovery Mechanisms

In one embodiment, the distributed computing environment may include a service discovery mechanism that provides methods for clients to find services and to negotiate the rights to use some or all of a service's capabilities. Note that a space is an example of a service. The service discovery mechanism may be secure, and may track and match outgoing client requests with incoming service responses.

A service discovery mechanism may provide various capabilities including, but not limited to:

- Finding a service using flexible search criteria.
- Requesting an authorization mechanism, for example, an authentication credential, that may convey to the client the right to use the entire set or a subset of the entire set of a service's capabilities.
- Requesting a credential, document, or other object that may convey to the client the service's interface. In one embodiment, the service's interface may include interfaces to a requested set of the service's capabilities.
- The tracking of discovery responses to the original requests. In one embodiment, each client request may include a collection of data that may also be returned in matching responses, thus allowing the requests and responses to be correlated.

In one embodiment of the distributed computing environment, a service discovery mechanism may provide a flexible search criteria based upon an extensible grammar. In one embodiment, a service name, service type, and other elements, if any, being searched for may be matched with elements in an XML document. In one embodiment, the XML document is the service advertisement for the service. XML may provide a flexible, extensible grammar for searching. XML also may provide type safety for matching elements. In one embodiment, the service names and service types may be type checked with the element types in the XML service advertisement.

In one embodiment, a distributed computing environment may include a mechanism for clients to negotiate service access rights. In one embodiment, the mechanism may be used to negotiate for a subset of a service's full capabilities. The result of the negotiation may be an authorization such as an authentication credential that conveys to the client the right to use the requested subset of the service's capabilities.

In one embodiment, the service discovery mechanism may allow a client to request a security capability credential from a service. In one embodiment, the client may present to the service a set of desired capabilities in

choosing services to access by using the string or XML document to pass custom search information between a client and service that may only be understood by the client and service.

Matching Component (Service) Interfaces

The distributed computing environment may provide a mechanism for matching a component (for example, a service) specification interface with a requested interface. For example, a client (which may be a service) may desire a service that meets a set of interface requirements. Each component may have a description of the interface to which it conforms. The specification interface matching mechanism may allow a component that best matches a requestor's interface requirements to be located. The specification interface matching mechanism may also allow for "fuzzy" matching of interface requirements. In other words, the mechanism may allow matching without requiring the exact specification of all aspects of the interface, thus providing a nearest match (fuzzy) mechanism. In one embodiment, the specification interface matching mechanism may be implemented as a multi-level, sub-classing model rather than requiring specification at a single interface level.

In one embodiment, a component may use an XML Schema Definition Language (XSDL) to describe its interface. XSDL may provide a human-interpretable language for describing the interface, simplifying activities requiring human intervention such as debugging. In one embodiment, the interface description may be provided as part of an advertisement (for example, a service advertisement) as described elsewhere in this document.

Using the specification interface matching mechanism, a basic desired interface may be compared to a set of component' interface descriptions. One or more components matching the basic desired interface may be identified. The interface descriptions may include subclass descriptions describing more specifically the interfaces provided by the components. In the search process, the class type hierarchy may be examined to determine if a given class is a subclass of the search type. In one embodiment, subclasses may inherit properties of the base class, and thus the subclass-specific information may not be examined in this phase. Thus, the search may be performed generically. The identified components may be searched at the next (subclass) level. The search may become specific to the subclass and may be performed by interpreting the subclass information included in the interface description. The search may continue through one or more subclasses until one or more components is determined which may provide the nearest match to the requestor's desired interface.

In one embodiment, an interface matching mechanism may provide the ability to distinguish among two or more components that implement similar interfaces. In one embodiment, the interface matching mechanism may provide the ability to distinguish among different revisions of the same component.

In one embodiment, a component description may be provided that includes a specification of the interface to which the component conforms. The component description may also include information about the component itself. The interface description and/or the component information may be used to differentiate among different implementations of a given interface. The component descriptions may include a canonical identifier and version information. The version information may allow component revisions to be distinguished. In one embodiment, the component description may be provided as part of an advertisement (for example, a service advertisement) as described elsewhere in this document.

In one embodiment, components may be searched for a particular canonical identifier. Two or more components may be identified with matching canonical identifiers. One or more components may be selected

access the service. Both the URI and schema may be provided in XML as an advertisement in a space. Thus, a mechanism for addressing and accessing a service in a distributed computing environment may be published as an advertisement in a space. Clients may discover a space and then lookup individual advertisement for services or content.

Figure 16 illustrates advertisement structure according to one embodiment. An advertisement 500, like other XML documents, may include a series of hierarchically arranged elements 502. Each element 502 may include its data or additional elements. An element may also have attributes 504. Attributes may be name-value string pairs. Attributes may store meta-data, which may facilitate describing the data within the element.

In some embodiments, an advertisement may exist in different distinct states. One such state may be a drafted state. In one embodiment, advertisements may initially be constructed in a drafted state that exists outside the bounds of a space. The creator of an advertisement may construct it in a variety of ways, including using an XML editor. Access to elements and attributes in the drafted state may be at the raw data and meta-data levels using any suitable means. Typically, events are not produced for changes made to advertisements in the drafted state. Therefore, the creator of the advertisement may be free to add, change, or delete elements as well as to achieve the desired attribute set, and then publish the advertisement for the rest of the distributed computing environment to see.

In one embodiment, another possible state for advertisements is a published state. Advertisements may move to the published state when inserted into a space. Once the advertisement is in a space, interested clients, and services may locate it, e.g. using its name and/or its elements as search criteria. For example, search criteria may be specified as an XML template document that may be compared (e.g. by the space service) with the advertisements in the space. Published advertisements may represent "on-line" services ready for clients to use. The message address (URI) of the service may be stored as an element in the advertisement. Advertisements that are removed from the space may transition back to the drafted state where they may be discarded or held. Removal may generate an event so interested listeners may be made aware of the change. Message gates are typically created from published advertisements.

In one embodiment, yet another possible state for advertisements is a persistent archived state. An archival procedure may turn a live published advertisement into a stream of bytes that may be persistently stored for later reconstruction. Archived advertisements may be sent (e.g. in their raw XML form) from the space to an archival service. The URI for an advertisement's archival service may be stored as an element in the advertisement. XML may provide a format for storing and retrieving advertisements and representing the state of advertisement elements sufficient to reconstruct the advertisement object(s). Advertisements may be stored in other formats as well, depending on archival service implementation. The process of making a published advertisement persistent may prepare the advertisement for the persistent archived state. Persistent advertisements may be stored (e.g. by an archival service) for future use in a persistent storage location such as a file or a database. A space through the archival procedure may enable advertisements to be stored, however the space does not necessarily play a role in how persisted advertisement entries are actually stored. How persisted advertisements are stored may be determined by the advertisement's archival service. Typically, no events are generated on behalf of archived advertisements. Also, changes may not be allowed for advertisements in the persistent archived state.

advertisement within an assumed space. In one embodiment, the syntax rules governing the construction of pathnames is that of the URI (Uniform Resource Identifier). In that embodiment, advertisement and space names therefore may not contain any URI reserved characters or sequences of characters. Pathnames to elements and attributes may also be specified using a URI. In general, element and attribute names may be appended to the pathname of an advertisement, such as:

http://java.sun.com/spacename/advertisement/element/attribute.

In one embodiment, the distributed computing environment may include a mechanism that allows a client to discover the URI of a space but restricts access to the service advertisement for the space. In one embodiment, rather than returning the full advertisement to the space, the URI of the space and the URI of an authentication service for the space may be returned. In order for the client to access the documents or services advertised in the space, the client first may authenticate itself to the authentication service at the URI provided in the return message. The authentication service may then return an authentication credential that may allow the client partial or full access to the space. When the client receives the authentication credential, the client may attempt to connect to the space to access the documents or service advertisements in the space.

The distributed computing environment may provide a mechanism or mechanisms that may enable a client to connect to a space. Embodiments of a connection mechanism may provide for client-space addressing, client authorization, security, leasing, client capabilities determination, and client-space connection management. A client-space connection may be referred to as a session. In one embodiment, a session may be assigned a unique session identification number (session ID). The session ID may uniquely identify a client-space connection. In one embodiment, a session lease mechanism may be used to transparently garbage collect the session if the client does not renew the lease.

The following is an example of using such a connection mechanism according to one embodiment. A client may obtain an authentication credential. In one embodiment, the space may provide an authentication service in response to a client's request for access to the space. The client may obtain the authentication credential through the authentication service. When the client receives the authentication credential, the client may initiate a connection to the space by sending a connection request message. In one embodiment, the connection request message may include the URI address of the space service, the authentication credential for the client and information about the connection lease the client is requesting. After the space receives the connection request message, the space may validate the message. In one embodiment, an XML schema may be used to validate the message. The client may then be authenticated using the authentication credential. In one embodiment, the information received in the connection request message may be used to determine the capabilities of the client to use the space. In one embodiment, each client of a space may be assigned its own set of capabilities for using the space. In one embodiment, an access control list (ACL) that may include capability information about one or more clients of the space may be used in client capabilities determination. In one embodiment, the information received in the connection request message may be used to look up the client's capabilities in the ACL.

After authenticating the client and determining the client's capabilities, the connection lease to grant the client may be determined. After the lease is determined, the structure for maintaining the client-space connection may be generated. A session ID for the connection may be generated. In one embodiment, each client-space connection may be assigned a unique session ID. In one embodiment, an activation space may be created and

space (e.g. obtained from keywords of the space), and information that can be used to set up a TCP connection, for example, with each space manager to perform operations on the respective space. Since the requesting device may receive responses from more than one space manager (or multiple space listings from a listener agent), this information may help the client select which space it wishes to connect to.

In addition to the multicast discovery described above, the discovery service may also perform discovery using unicast messaging (e.g. over TCP) that can be used to discover a space manager at a known address on the network (e.g. the Internet, other WAN, LAN, etc). The unicast discovery message may include a request for a space service at a known URI to provide its service advertisement. The multicast and unicast discovery protocols are defined at the message level, and thus may be used regardless of whether the devices participating in the discovery support Java or any other particular language.

The discovery protocol may facilitate the proliferation of clients independently of the proliferation of server content that supports those clients within the distributed computing environment. For example, a mobile client may have its initial default space built into its local platform. In addition to local services advertised in the default space, the mobile client may have services that search for additional spaces, such as a service to access the discovery protocol or a service to access space search engines.

In one embodiment, the distributed computing environment space discovery protocol may define a set of XML messages and their responses that may allow clients to:

- Broadcast protocol-defined space discovery messages on their network interfaces.
- Receive from listeners XML messages describing candidate spaces that those listeners represent.
- Select one of those discovered spaces as default, without the client needing to know the address of the selected space.
- Obtain information on the selected space, such as its address, so the client may later find that same space via means outside of the discovery protocol (useful if later the client wants to access a space which is no longer local, but which still is of interest to the client).

In some embodiments, the multicast and unicast discovery protocols may require an IP network. Although these discovery protocols meet the needs of devices that are IP network capable, there are many devices that may not be directly supported by these discovery protocols. To meet the needs of such devices in discovering spaces in the distributed computing environment, a pre-discovery protocol may used to find an IP network capable agent. The pre-discovery protocol may include the device sending a message on a non-IP network interface requesting a network agent. The network agent may set up a connection between itself and the device. Once the connection between device and agent is set up, the agent participates in the discovery protocol on IP networks on behalf of the device for which it serves as agent. The network agent may also provide an interface for the device to the distributed computing environment in general. For example, gates may be constructed in the agent on behalf of the device for running services advertised in discovered spaces. See the ***Bridging*** section.

Another way that clients may locate spaces in the distributed computing environment is by advertisement of a space in another space. A space is a service, thereforeso, like any other service, it can be advertised in another space. As shown in Figure 18, a client 200b may find an advertisement 206 in a first space 204a for a second space 204b. Space 204b may in turn include advertisements to additional spaces. Because a service

as indicated at 300. The authentication service may be specified in the service advertisement of the space service. The client of the space uses the authentication credential, the XML schema of the space (from space's service advertisement), and the URI of the space (from space's service advertisement) to construct a gate for the space, as indicated at 302. The client of the space may then run the space service by using the gate to send messages to the space service. A first such message is indicated at 304.

For embodiments employing authentication, when the space service receives the first message from the client, with the authentication credential embedded, the space service uses the same authentication service (specified in the service advertisement of the space service) to authenticate the client, thus establishing its identity, as indicated at 306. The space service may determine the client's capabilities and bind them to the authentication credential, as indicated at 308.

As indicated at 310, a client of a space may run various space facilities by sending messages to the space service. In one embodiment, when a client of a space sends a request to the space service, it passes its authentication credential in that request, so the space service can check the request against the client's specific capabilities.

Each space is typically a service and may have an XML schema defining the core functionality of the space service. The XML schema may specify the client interface to the space service. In one embodiment, all space services may provide a base-level of space-related messages. The base-level space functionality may be the basic space functionality that is capable of being used by most clients, including small devices such as PDAs. It may be desirable to provide for additional functionality, e.g. for more advanced clients. Extensions to the base-level space may be accomplished by adding more messages to the XML schema that advertises the space. For example, in one embodiment, the base-level messages do not impose any relationship graph upon the advertisements. Messages, for example, to traverse a hierarchy of advertisements may be a space extension. Such additional functionality may be provided through one or more extended XML space schemas or schema extensions for a space. The extended schemas may include the base schema so that clients of an extended space may still access the space as a base space.

In one embodiment, a base space service may provide a transient repository of XML documents (e.g. advertisements of services, results of running services). However, a base space service in one embodiment may not provide for advanced facilities to support persistence of space content, navigation or creation of space structure (e.g. hierarchy), and a transactional model. A mechanism for supporting persistence, hierarchy, and/or transactions is by extending the XML schema. Since extended spaces still include the base XML schema, clients may still treat extended spaces as base spaces, when just the base space functionality is all that is need or all that can be supported.

In one embodiment, the base space may be transient. The base space may be acceptable for many purposes. Service providers may register their services in various spaces. In one embodiment, services must continuously renew leases on the publishing of information in the spaces. By this nature, the services advertisements may be transient in that they may often be rebuilt and/or reconfirmed. However, it may be desirable to provide for some persistence in a space. For example, a space that has results may provide some persistence for users that want to be sure that results are not lost for some time. In one embodiment, persistence may be provided for by specifying a space interface where the client may control which objects in the space are

space may also provide a look-up facility that allows a client to search for a service by providing keywords or string names. In one embodiment, a space facility may provide a mechanism to look up a space entry that has been added to the space. The look up facility may search by string to match for name, or wildcard, or even database query. The look up facility may return multiple entries from which the client may select one or perform a further narrowing search. In one embodiment, the look-up facility may provide a mechanism to locate a service advertisement matching a particular XML schema. The client may indicate a particular XML schema, or part of a particular XML, to be searched for within the space. Thus, a service may be searched for within a space according to its interface functionality.

Another space facility that may be provided in the distributed computing environment is a mechanism that allows services and clients to find transient documents based upon a typing model such as XML. The mechanism may be a general-purpose, typed document lookup mechanism. In one embodiment, the lookup mechanism may be based upon XML. The lookup mechanism may allow clients and services to find documents in general, including services through service advertisements.

In one embodiment, a space lookup and response message pair may be used to allow clients and services to find XML documents stored within a network transient document store (space). The space may be a document space used to store a variety of documents. In one embodiment, the documents are XML documents or non-XML documents encapsulated in XML. Spaces are further described elsewhere herein. The lookup messages may work on any kind of XML document stored in the space, including service advertisements and device driver advertisements. In one embodiment, a client (which may be another service) may use a discovery mechanism as described elsewhere to find one or more document spaces. Then, the client may use space lookup messages to locate documents stored in the space.

The distributed computing environment may include a mechanism that allows services and clients to subscribe to and receive events about the publication of XML documents. Events may include the publication of and removal of XML documents to and from a transient XML document repository such as a space. In one embodiment, an event may be an XML document that refers to another XML document.

In one embodiment, a space event subscription and response message pair may be used to allow clients and services to subscribe for events regarding documents that are added to or removed from a space. In one embodiment, an event subscription may be leased using the leasing mechanisms described elsewhere herein. In one embodiment, a subscription may be cancelled when the lease is cancelled or expires. In one embodiment, renewing the lease to the subscription may renew a subscription.

In one embodiment, an event subscription message may include an XML schema that may be used as a document matching mechanism. Documents that match the schema may be covered by the subscription. In one embodiment, any document added to a space and that matches the XML schema may generate a space event message.

A space facility may also be provided to which a client may register (or unregister) to obtain notification when something is added to or removed from the space. A space may contain transient content, reflecting services that at added and removed from the space. A mechanism may be provided to notify a client when a service becomes available or becomes unavailable, for example. A client may register with an event service to obtain such notification. In one embodiment, a client may register to be notified when a service having a name matching a

computing environment. The client may then run the service using the constructed gate and XML messaging. The service may similarly construct a service gate for XML message communication with the client.

To summarize, an example use of a space is discussed as follows. A client may access (e.g., connect to) a space service. (A service may act as a client for the purpose of accessing or otherwise using a space.) The space service may store one or more service advertisements and/or other content in a space, and each of the service advertisements may include information which is usable to access and execute a corresponding service. The space service may include a schema which specifies one or more messages usable to invoke functions of the space service. For example, the schema may specify methods for reading advertisements from the space and publishing advertisements in the space. The schema and service advertisements may be expressed in an object representation language such as eXtensible Markup Language (XML). In accessing the space service, the client may send information such as an XML message (as specified in the schema) to the space service at an Internet address. In accessing the space service, the client may search the one or more service advertisements stored in the space. The client may select one of the service advertisements from the space. In one embodiment, the client may send an instantiation request to the space after selecting the desired service advertisements from the space. A lease may be obtained for the desired service, and the lease and the selected service advertisement may be sent by the space service to the client. The client may then construct a gate for access to the desired service. The desired service may be executed on behalf of the client.

Another facility provided by a space service may be the spawning or creation of an empty space. This space facility may allow a client (which may be a service to another client) to dynamically create a new space. In one embodiment, this space facility may include an interface for spawning an empty space with the same functionality (same XML schema or extended schema) as the space from which it is spawned. This facility may be useful for generating (e.g. dynamically) spaces for results. For example, a client may spawn a space a request a service to place results or advertise results in the spawned space. The client may pass the spawned space URI and/or authentication credential to the service. Or a service may spawn a space for results and pass the spawned space URI and/or authentication credential to the client. In some embodiments, once a space is spawned, it may be discovered just like other spaces using one or more of the space discovery mechanisms described herein.

By using a mechanism in which a space may be created via an interface in another space (e.g. a space spawning facility), new spaces may be created efficiently. For example, in one embodiment, storage for the spawned space may be allocated using the same facility used by the original space for storage. Also, a spawned space may share a common service facility with its original (or parent) space. For example, a new URI may be assigned to the new space. In one embodiment, the new URI may be a redirection to a common space facility shared with the original space. Thus, a newly spawned space may use the same or some of the same service code as that of the original space.

Space facilities may also include security administration, for example, to update the various security policies of the space, and other administrative facilities. For example, the number and age of advertisements may be controlled and monitored by a root space service. Old advertisements may be collected and disposed. See, e.g., the Leases section herein for when an advertisement may be considered old. The service implementing the space may be under the control of an administrator. The administrator may set policy in a service dependent manner. Space facilities may also include a facility to delete an empty space.

Thus, several different mechanisms may be employed within the distributed computing environment for a service to return results to a client. The actual results may be returned to the client by value in an XML message, or results may be returned to the client by reference with the actual results (or advertisement for the actual results) put in a space and the client receiving a message referencing the results in the space. Moreover, results, or results advertisements, may be placed in a space and the client notified by event.

Another mechanism for handling results may be for the client to specify another service for the results to be fed to. For example, when a client runs a service that will produce results, the client may instruct that service (e.g. through XML messaging) to send the results to another service for further processing. This may involve the client indicating the URI of an advertisement for the other service so that the result-producing service may generate a gate to the other service in order to run the other service and pass it the results. In this example, the result-producing service may be a client of the other service. In some embodiments, the client may send the schema or a pre-constructed gate to the result-producing service to access the service for further processing. An example of a service for further processing is a display service that may display the results for the original client. This display service may be on or associated with the same device as the client.

Result spaces and method gates may allow the distributed computing environment to provide a simple remote method invocation that is practical for thin clients with minimal memory footprints and minimal bandwidth, because it need not have the adverse side effects of huge program objects (along with needed classes) being returned (necessarily) across the network to the client as in conventional remote method invocation techniques. Instead, results may be returned to a result space, and only if desired (and if they can reside on the client) are the actual objects downloaded to the client.

The mechanism by which the distributed computing environment may provide for remote method invocation is as follows (refer also to the description of method gates in the Gates section herein). An object may be advertised (e.g. as a service or as part of a service) in a space. The advertisement includes a reference that contains the URI (e.g. URL) of the object, along with other access parameters, such as security credentials and XML schema. A client may have or may construct a client method gate for the object, which for every method of the object (or service) itself may have a wrapper method that takes the method parameters and creates a request XML message to invoke a method of the object. The XML message is sent to a service gate that invokes the actual method on the service object. When that method returns a result object, the service gate may post the result object in a results space, and may return a message to the client with a reference to the result object.

Thus, for a client to invoke a remote method, the client first sends a message to instantiate an object (e.g. service), such as described above. In one embodiment, instantiation of an object may include the creation or spawning of a results space. In another embodiment, results space creation may be independent from the object instantiation. Instantiation may return the object URI to the client, and the client and service gates may be dynamically created when a client requests instantiation. In some embodiments, a results space may already exist and be advertised by the object (service). Some part or all of the gates may also have been pre-constructed or reused.

Once a client has initiated an object, a local call of the appropriate client method gate will affect a remote call to the actual remote object, as described above. The remote method invocation approach of the distributed computing environment may be recursive, with object references returned to the client, instead of the objects itself,

For some devices that provide a service, the overhead of finding a space to advertise its service and maintain that advertisement is undesirable. In one embodiment, rather than searching for and maintaining a space or spaces to publish service advertisements, services on some devices may transmit their advertisements in response to connection requests. For example, a printer device with a printer service that is available on a proximity basis may not maintain an advertisement in a space (on the device or external to the device). Instead, when another device establishes a connection with the printer device (for example, a user with a laptop running a client desires to print a document), the printer service may transmit the service advertisement to provide the XML service schema for connecting to and running the service that provides printing functionality on the printer device. Also, some devices may only maintain advertisements for their services in a certain vicinity or local network. Such a device may not desire to support or may not have access to transports for broader accessibility.

One example of a service device in which it may be desirable for the device to avoid or limit maintaining service advertisements in a space is a device whose functionality is available on a proximity basis. Proximity-based services may provide advertisements of their functionality upon request. These advertisements may not be broadly accessible. For example, proximity-based services may be provided in a wireless communications system. The term "wireless" may refer to a communications, monitoring, or control system in which electromagnetic or acoustic waves carry a signal through atmospheric space rather than along a wire. In most wireless systems, radio frequency (RF) or infrared (IR) waves are used. Typically, in proximity-based wireless systems, a device comprising a transceiver must be within range (proximity) of another device to establish and maintain a communications channel. A device may be a hub to connect other devices to a wireless Local Area Network (LAN).

As mentioned, embodiments of the distributed computing environment may provide a mechanism using a lookup space that allows clients to rendezvous with services. In a proximity computing environment, one embodiment of the distributed computing environment may provide a service discovery mechanism that clients may use to discover services without using lookup spaces as rendezvous points. An example of a proximity computing environment is an IrDA point-to-point communications environment. In a proximity computing environment, the proximity mechanism may find the "physical" location of the service for the client. For example, in an IrDA environment, the client device may be physically pointed at the device including the service(s) that the client desires to use.

The proximity service discovery mechanism may enable the client to directly look for service advertisements rather than sending a lookup request to a lookup space to look for service advertisements. Since the client device may have established a proximity connection to the service device, the client may directly request the desired service. For example, a PDA client device may establish a proximity connection to a printer device; the client may "know" to request a printer service connection on the printer device.

In one embodiment, the client may send a proximity service discovery message to the service device. The message may include information that may specify a desired service on the service device to which the client device has a proximity connection. In one embodiment, a service on the service device may respond to the proximity service discovery message, and may send to the client the service advertisement that the client may use to connect to the desired service. The proximity service discovery message may also include information that may

the proximity devices. Publishing device 1404 may act as a bridge between the network 1412 and the proximity connections 1414 to the proximity-based devices.

Leases

Leases may be used in the distributed computing environment to deal with partial failure, resource synchronization (scheduling), and to provide an orderly resource clean-up process. Leases may help the overall distributed system manage independent clients and services that may come and go. The various resources that clients obtain from services (including space services) may be leased from those services. In general, not every resource can or needs to be leased. In one embodiment, it is up to the implementation of each particular service to determine which of its resources need to be leased. In particular, resources used by a large amount of clients simultaneously may not need leasing or instead may require custom leasing protocols. This class of leasing may be left to the service provider. Custom protocols, such as those to implement transactions for example, may be built upon the base leasing scheme. In one embodiment, the base leasing model is a relative time-based model.

Services may issue leases to clients and provide operations on those leases. In one embodiment, all such lease functionality of a service is part of that service's XML schema. Thus, a client may use its gate (corresponding to the service and constructed for the service's XML schema) to perform lease operations. In one embodiment, all services that issue leases provide the following lease operations (only allowed by the owner of the lease): (i) renewing a lease (parameters specified: lease (e.g. lease ID, lease credential), new lease time requested), and (ii) canceling a lease (parameter specified: lease (e.g. lease ID, lease credential)). In one embodiment, all leases are granted for a particular amount of relative time (duration of lease) that may be negotiated. The requestor may specify a certain amount of time (e.g. in seconds), and the grantor may grant the lease for any amount up to that time. In one embodiment, a -1 value may be used to specify an indefinite lease.

In one embodiment, a service advertisement may include one or more leasing addresses. In one embodiment, the leasing addresses may be URIs. Standard leasing messages to renew and cancel service resource leases may be sent to a leasing URI. An example lease URI:

<leaser>service1://resource1</leaser>

An advertisement may also include various leasing messages as described above. Leasing messages may include messages to renew and cancel leases for resources of the service. In one embodiment, the messages may be comprised in an XML schema in the advertisement.

The leasing mechanism may provide a mechanism to detect service and client failure. Leases may also provide a mechanism to provide shared and exclusive resource access. In one embodiment, all service resources either have no lease (resource is not leased and therefore available), a shared lease (resource accessed by multiple clients), or an exclusive lease (resource is accessed by exactly one client at a time). In one embodiment, all resources begin in the no lease state. A no lease state signifies there is no current access to the underlying resource, and indicates that there is an interest in the resource remaining in existence and thus available for leasing. The leasing level may be increased from none to shared, none to exclusive, or shared to exclusive. Lease isolation levels may also be decreased from exclusive to shared, exclusive to none, and shared to none. In one embodiment, clients may voluntarily increase or decrease the lease isolation level, or may be requested by the service to do so. A response message from the service may indicate if the isolation level change was accepted.

authorized, credentialed client (and the service issuing the lease) from canceling the lease. See the "Authentication and Security" section for more detail.

Figure 44 is a flow diagram illustrating a mechanism for leasing resources. In step 2000, a client may request a lease on a resource provided by a service. In one embodiment, leases may be time-based, i.e. granted for a period to the client. In another embodiment, leases may be non-time-based leases that are maintained until cancelled by the client or the service. In one embodiment, the service may be a space service, and the resource may be a service advertisement for another service and leased to the client by the space service. In one embodiment, the service may be a space service, the client may itself be a service, and the resource may be the publishing of a service advertisement by the space service on behalf of the client/service. In one embodiment, the client may send a request message to the service that specified the resource and requests the lease on the resource. In one embodiment, the message may specify a requested lease period. In one embodiment, a client message gate may send the lease request message to the service on behalf of the client. In one embodiment, the lease request message may be an XML message.

In step 2002, the service may grant the lease on the resource to the client. In one embodiment, the service may receive a lease request message from the client. The service may then grant the lease on the resource. In one embodiment, the lease request message may include a requested lease period. In one embodiment, the service may grant the lease for a granted lease period less than or equal to the requested lease period. In one embodiment, the requested lease period may be for an indefinite lease (i.e. the period does not expire). In this embodiment, the client or service must cancel the lease, as the lease itself does not expire.

The service advertisement provided to the client may be used in establishing an initial lease of resource(s) provided by the service and requested by the client during access of the service. In one embodiment, the lease on the resource may be granted at service instantiation time. In this embodiment, the client may not send a lease request message to the service requesting the lease. In this embodiment, the service may grant an initial lease on the resource or resources to the client when instantiated from the service advertisement. For example, a space service may, in response to a request message from a client, instantiate a service from a service advertisement on the space service. The service may grant the client leases on one or more resources provided by the service when instantiated without requiring the client to explicitly send a message requesting the leases.

In step 2004, the client may request renewal of the lease. In one embodiment, the service may send a lease renewal request message to the client. In one embodiment, the lease renewal request message may be sent to the client prior to expiration of a previously granted lease period. The client may respond to the message with a lease renewal response message to the service requesting renewal of the lease. In one embodiment using time-based leasing, the lease renewal response message may include a requested lease period. In another embodiment using non-time-based leasing, the lease renewal response message may request that the lease be continued. In one embodiment, the client may return a lease renewal response message that instructs the service that the lease is no longer required. In one embodiment, the client may not send a lease renewal response message, and the service might assume, when no response message is received by the service, that the client no longer requires the lease. For example, the client may have been disconnected from the network. This provides the service with a mechanism for detecting non-used leases and for garbage collecting resources including leasing resources.

become unavailable. In one embodiment, the service may send a message to the client to inform the client if the lease has been cancelled.

In one embodiment, a security credential, such as an authentication credential described below, may be included in lease messages sent between the client and service as described in Figure 44. For example, the client may embed the security credential in lease renewal and lease cancel request messages sent to the service. The service, upon receiving a message, may examine the credential to verify it is from the leaseholder, and thus to verify the authenticity of the message. This provides security in the leasing mechanism by preventing entities not having the proper credentials from modifying the client's current lease(s) with the service.

In one embodiment, the lease messages sent between the client and the service may all be in a data representation language. In one embodiment, the data representation language may be eXtensible Markup Language (XML). In one embodiment, all messages may be sent and received by a service message gate and a client message gate. In one embodiment, the lease messages may be described in an XML message schema provided to the client and used to construct the client message gate. In one embodiment, the XML message schema may have been acquired by the client in a service advertisement, for example, retrieved from a space service.

The leasing mechanism may also provide a mechanism to detect stale advertisements. When a service publishes its advertisement in a space, that service obtains a lease on this publishing of its advertisement. Each advertisement may contain a time by which the service promises to renew the advertisement. In one embodiment, all time-out values are specified in seconds. If the service continues to renew its lease, the space is provided some assurance that the service advertised is still being offered. The renewal time may be counted down towards zero by the space service. If the service does not renew its lease, the service may have failed, or it may no longer wish to, or be able to provide the service. When the lease is not renewed, the space service marks the service advertisement stale, so it does not make it available to clients. Services renew advertisements by sending a renewal message to the space. The space service receives these messages and resets the advertisement renewal time back to its initial value.

In one embodiment, stale advertisements are not automatically deleted. Depending upon the policies of the space, it may choose to delete stale service advertisements that have expired for a reasonably long period of time. The deletion policy may be set by the space service. The space service may search for stale advertisements and either delete them or bring them to the attention of an administrator, for example.

A space service may use leases to manage the resources its facilities provide to clients (including other services) of the space. For example, when a client desires to use a service, the space service may request a lease for the client as part of service instantiation. Service instantiation may be performed to allow a client to run a service. To instantiate a service, a client may first select one of the service advertisements published in a space. The client may use the various facilities provided by the space to look up advertisements in the space. Then the client may request the space to instantiate the service. The lease acquired during service instantiation is on use of the service advertisement (not the same as the lease on publishing of the service advertisement). It should be noted that the space service may allow multiple clients to have a lease on use of a service advertisement if the

the specific computer that the resource is housed in and the specific name of the resource (typically a file name) on the computer.

Clients and services (both may be implemented on devices as software and/or firmware) may be connected over the Internet, a corporate intranet, a dynamic proximity network, within a single computer, or by other network connection models. The size and complexity of the devices supporting clients and services may range, for example, from a simple light switch to a complex, highly available server. Example devices include, but are not limited to: PDAs; cellular phones; notebook, laptop, and more powerful PCs; and more powerful computer systems, up to and including supercomputers. In some embodiments, the distance, latency, and implementation of clients and services may be abstracted, with a common discovery and communication methodology, creating a "black box" effect. This definition approach allows software implementation issues to be dealt with by the underlying platform, yielding a loosely coupled system that may be scaled to Internet proportions.

The distributed computing environment may provide an Internet-centric programming model including WEB and XML content representation, dynamic device discovery, and secure device communication that is accessible from a wide range of network devices. The distributed computing environment may include a network-programming model abstracted above the CPU level. The programming model may include the following properties:

- URI addresses
- Strongly typed data called content (addressed with URIs)
- Substantially unlimited amount of persistent content storage (e.g. stores), (containing XML and non-XML content, such as that identified by MIME types)
- Substantially unlimited amount of transient content memory called spaces (containing XML content)
- Descriptive XML metadata (data about data) content advertisements that may be stored in a space to notify interested clients.
- A substantially unlimited number of instructions (embodied as messages)
- Secure message endpoints (gates) addressed by URIs
- Data flow support (event messages) to coordinate work flow between distributed software programs

Services and clients may run as programs within the distributed computing environment. Services may advertise their capabilities to clients wishing to use the service. Clients may or may not reside within the same network device, and that device's code execution environment may or may not support the Java platform.

Using URIs to address content and message endpoints gives the distributed computing environment a powerful addressing scheme. The address may specify the location of the content or endpoint, and may specify the route (or transport protocol) to be used. Items addressed using URIs also may have an associated security credential. The security credential may be used to control what clients are allowed access to the item, as well as which operations authorized clients are allowed to perform on that item.

The high degree of access provided by the distributed computing environment may be controlled by appropriate authentication and security systems and methods. Authentication and security in the distributed computing environment may include, but are not limited to: verifying the typing correctness of XML content in a message; securely identifying the sender to the receiver; a mechanism to check the integrity of messages sent from a client to a service and vice versa; and a mechanism of describing a service's set of accepted messages to a client

authentication credential in every message sent from the client to the service. The messages may be received by the service message gate and then checked by the authentication service to ensure that the message is from the client and that the message request is within the capabilities of the client. In another embodiment, the service message gate may handle capability determination and message checking for capabilities without using the authentication service.

The client and service message gates may work together to provide a secure and reliable message channel. The gates may serve as secure message endpoints that allow the client to run the service by sending and receiving secured, authorized XML messages to and from the service.

Operations in the distributed computing environment may be embodied as XML messages sent between clients and services. The protocol used to connect clients with services, and to address content in spaces and stores, may be defined by the messages that can be sent between the clients and services. The use of messages to define a protocol may enable many different kinds of devices to participate in the protocol. Each device may be free to implement the protocol in a manner best suited to its abilities and role.

A service's capabilities may be expressed in terms of the messages the service accepts. A service's message set may be defined using an XML schema. An XML message schema may define each message format using XML typed tags. The tag usage rules may also be defined in the schema. The message schema may be a component of an XML advertisement along with the service's message endpoint (gate) used to receive messages. Extensions (more capabilities) may be added to services by adding messages to the XML message schema.

In the distributed computing environment, authorized clients may be able to use all of a service's capabilities, or may be limited to using a subset of the service's capabilities. In one embodiment, once a set of capabilities has been given to a client, the client may not change that set without proper authorization. This model of capability definition may allow for services levels that run from a base set of capabilities to an extended set.

Service instantiation may be performed to allow a client to run a service. To instantiate a service, a client may first select one of the service advertisements published in a space. The client may use the various facilities provided by the space to look up advertisements in the space. Then the client may request the space to instantiate the service. Service instantiation may include, but is not limited to, the following steps:

1. Client requests space service to instantiate a service.
2. Space service verifies client is allowed to instantiate the service.
3. Space service obtains a lease on the service advertisement for the client with the lease request time specified by the client. Alternatively, the service advertisement may be provided to the client without using the leasing mechanism.
4. Space service sends a message to the client that includes the lease allocated in steps 3, and the service advertisement of the service.
5. Client runs the authentication service specified in the service advertisement, and obtains an authentication credential.
6. Client constructs a client message gate for communicating with the service.

In order to provide trust between clients and services in the distributed computing environment, a series of dynamically generated numbers (keys, or tokens) may be used as security or authentication credentials for

In some embodiments, a mechanism for verifying that a client attempting to run a service is an authorized client, for verifying that the service advertisement received by the client is an authorized service advertisement, and/or for verifying that the space from which the client received the service advertisement is authorized may be based upon a public key/private key asymmetric cryptographic mechanism. In this mechanism, an authorized sending entity may embed a public key in a message and encrypt the message including the public key with its private key. An entity receiving the encrypted message may decrypt the message using the public key and find the same public key embedded in the decrypted message, and thus verify that the message is from the authorized entity, since only that entity has the private key necessary to encrypt the message. Thus, an entity may issue a credential that is substantially unforgeable, and that other entities may decrypt (with the appropriate public key) to verify messages sent by the entity.

In one embodiment, messages in the distributed computing environment may include several layers of encrypted public keys for authorizing the entities in the client-service communications model. In one embodiment, a service may build a credential C1 including its public key X1. The service may then encrypt the credential using its private key Y1. The service may include the encrypted credential C1 in a space as part of its service advertisement. The space then may build a new credential C2 including credential C1 and the space's public key X2. The space may then encrypt the credential C2 with its private key Y2. When a client requests the service from the space, the space sends the service advertisement and credential C2 to the client. When the client constructs a gate, it has the gate send messages to the service including another credential C3 which includes encrypted credential C2 and the client's public key X3. C3 may be encrypted using the client's private key Y3 prior to sending. The client's public key X3 and the space's public key X2 may also be sent with the messages. In one embodiment, the space's public key X2 may be included in C3, and thus encrypted in C3. In this embodiment, the client's public key X3 may be sent unencrypted in the messages. In another embodiment, the client's public key X3 and the space's public key X2 may be sent unencrypted in the messages.

After the gates are created, in one embodiment, the client may send a message including encrypted credential C3 and public key X3 to the service. In one embodiment, the space's public key X2 may be included in C3. In another embodiment, public key X2 may be included (unencrypted) in the message external to C3. When the service receives the message, it may decrypt credential C3 using the received client's public key X3 and check the public key X3 embedded in the decrypted credential C3 against the received public key X3 used to decrypt the message, thus verifying the message is from an authorized client. The service may then decrypt credential C2 (extracted from decrypted credential C3) with the space's public key X2, and check the public key X2 embedded in the decrypted credential C2 against the space's public key X2, thus verifying that the space credential C2 came from an authorized space. In one embodiment, the space's public key X2 may have been included in credential C3. In another embodiment, the space's public key X2 may have been included in the message, external to C3. In yet another embodiment, the service may obtain the space's public key X2 elsewhere, for example, from the space itself. Finally, the service may decrypt credential C1 (extracted from decrypted credential C2) with the service's public key X1, and check the decrypted credential to verify that the credential was created by the service (the credential should include the public key of the service).

advertisement to authenticate the client, and thus may establish a binding of the authentication credential to the identity of the client.

As previously discussed, some results produced by a service may be advertised in a space and ultimately accessed using a results gate. The results gate may or may not contain the same security credential as the input gate used to generate the results. Because input to a service may be asynchronous from its output (the results), the results may have a different set of access rights associated with it. For example, a payroll service may allow a different set of clients to initiate payroll than to read the payroll service's results (paychecks). Thus, a client may have to go through a separate authentication process to obtain access rights to the results, which may include receiving an authentication credential for the results from an authentication service specified in an advertisement for the results.

Message gates may offload most security checks from services. Services may focus on providing capability and authenticating clients. A principle of least privilege may be supported by giving clients access to only those capabilities that are requested (or assigned).

Security checks may occur when a gate is created and/or when a gate is used (when messages are sent and/or received). When a client requests access to an advertised item (service), the process of gate creation may begin. During this process, the client gate factory may work with the service to mutually authenticate each other. The checks performed at gate creation time may be extensive, and may minimize the number of checks performed during gate usage. After the service has authenticated the client, the service may determine specific capabilities for the client (e.g. what the client is allowed to do on the service), and associate the capabilities with the client's authentication credential. These specific capabilities may specify what operations the client is allowed to perform on the service. Since the gates may ensure that every message contains the authentication credential, the service can then check each request when it is received against the capabilities of the authenticated client.

Gate creation checks may ensure that a client has permission to use some or all of the service capabilities designated by the XML message schema. In one embodiment, these checks may be implemented using access control lists (ACLs) in conjunction with an authentication service such as Kerberos. A challenge-response sequence (such as a password) may also be used to authenticate a client. In some embodiments, a hardware-based physical identification method may be used to authenticate the client. For example, the user may supply a physical identification such as a smart card for identification and authorization. Other mechanisms for authentication may be used in other embodiments.

In one embodiment, whatever means is used to authenticate the client, the authentication may be invisible to both the client and service, the gate factory may be aware of which authentication service to use, and the authentication service handles the authentication mechanism and policies. Gate factories may be product and environment dependent, or may even be controlled by a configuration management system. In one embodiment, the degree and method of client isolation may be platform dependent, but is known to the gate factory. In some embodiments, a hardware-based physical identification method may be used to authenticate the client. For example, the user may supply a physical identification such as a smart card for identification and authorization. Other mechanisms for authentication may be used in other embodiments.

Message gates in the distributed computing environment are typically associated with a single client. The gate factory may determine the means of association. The checks performed at message send time may ensure that

service are using that spawned space to store results, for example, if the client and service desire to keep the results private.

After running a service, the client may change the authentication policies of the spawned space using a security administration space facility, and other clients or services may then access the spawned space. In addition, the spawned space's service advertisement may be made available to other clients of the spawned space (the other clients may be services) using the discovery protocol or other means.

The message transport layer in a distributed computing environment may include mechanisms for protecting the security and integrity of communications among clients and services during transport. This security may be referred to as "wire security" or "transport security" to distinguish it from the authentication security implemented by the messaging system including gates. Encryption of messages may be provided at the message transport layer of the distributed computing environment. Services that request an encrypted transport may do so by tagging the XML advertisement. The gate factory may then create a gate (or gates) that uses a secure message transport such as those provided by Bluetooth and HTTPS.

HTTPS (Secure Hypertext Transfer Protocol) is a Web protocol that encrypts and decrypts user page requests as well as the pages that are returned by the Web server. HTTPS may use a multi-bit key size (may vary from 40 to 128- bit or more) for a stream encryption algorithm (e.g. RC4), to provide an adequate degree of encryption for commercial exchange. HTTPS may be used as a transport in the distributed computing environment.

Bluetooth is an emerging peer-to-peer wireless communications standard. The Bluetooth key generation algorithms may be used in the distributed computing environment. Bluetooth may support encryption keys. Encryption keys are transport dependent, while client, service, and combination keys may be transport independent.

Figure 26a - An authentication service providing an authentication credential to a client

Figure 26a is a flow diagram illustrating an authentication service providing an authentication credential to a client according to one embodiment. A client in the distributed computing environment may desire a service to perform one or more functions on behalf of the client. In one embodiment, an authentication service may be provided for use by the client and the service when setting up a secure messaging channel. An authentication service may perform functions for the client and/or service including authenticating the client and/or service and negotiating the desired level of security and the set of messages that may be passed between the client and service. The authentication service may be a process that is executing within the distributed computing environment. The authentication service may be executing on the same device as the service and/or the client, or alternatively the authentication service may be executing on a separate device such as an authentication server. In one embodiment, the authentication service may be an Internet-based service. The authentication service may have its own address, for example, a Universal Resource Identifier (URI), through which the client and/or service may communicate with the authentication service. In one embodiment, the address of the authentication service may be provided to the client in the service advertisement for the service. The client and service sharing an authentication service may help insure that a secure messaging channel may be established between the client and the service, as any of several security and authentication protocols may be used in the messaging channel.

Figure 26b is a flow diagram expanding on step 1002 of Figure 26a and illustrating an authentication service generating an authentication credential according to one embodiment. In step 1002a, in one embodiment, the authentication service may obtain a client token and a service token. In another embodiment, the authentication service may obtain only a client token. In one embodiment, the client token may be a unique identifier for the client in the distributed computing environment. In one embodiment, the service token may be a unique identifier for the service in the distributed computing environment. For example, the public keys from a public/private key encryption mechanism may be used as unique identifiers for the client and the service. In one embodiment, the client may receive the service token in the service advertisement, and the client may provide the client token and the service token to the authentication service. In another embodiment, the client may provide the client token and the service advertisement URI to the authentication service, and the authentication service may retrieve the service token from the service advertisement.

In step 1002b, the authentication service may verify the client and/or the service. In one embodiment, the authentication service may use the client token and the service token obtained in step 1002a to verify the client and/or service. In another embodiment, only a client token was obtained in step 1002a, and thus only the client token is used to verify the client in step 1002b. In one embodiment, the client may have previously registered its client token with the authentication service, and the authentication service may compare the received client token to the registered client token to verify the client as a valid client. In one embodiment, the client may access the authentication service using a challenge/response mechanism such as a logon account with password and thus may be verified as a valid client. In one embodiment, the service may have previously registered with the authentication service, and may have provided its service token to the authentication service. The authentication service may then verify that the client is attempting to access a valid service by comparing the received service token to the previously registered service token. Other types of client and service authentication may also be used. For example, the client may provide a digital signature or digital certificate that the authentication service may use to authenticate the client and/or to authenticate the service the client is trying to access.

In step 1002c, the authentication service may generate an authentication credential. In one embodiment, the authentication credential may include an authentication token that only the authentication service can create. In one embodiment, the authentication service may use the client token and the service token in generating the authentication credential. In another embodiment, the authentication service may use just the client token to generate the authentication credential. In yet another embodiment, the authentication service may not use an obtained token in the generation of the authentication credential, but may instead use an authentication credential generation algorithm to generate a substantially unforgeable authentication credential. In one embodiment, the authentication service may combine the service token and client token to create a unique authentication credential. For example, the service token and client token may be 64-bit values, and the two tokens may be combined to generate a 128-bit authentication credential. Other embodiments may use other methods to generate an authentication credential.

Figure 41 - Creating a gate

Figure 41 is a flow diagram illustrating the creation of a gate for a client according to one embodiment. In one embodiment, a gate factory may be trusted code on the client for generating gates based on XML service

schema into the client message gate, thus restricting the client's access to the service. In one embodiment, the authentication service may determine a subset of the total set of messages that the client may send to the service. One or more levels of access may be provided for a service in the distributed computing environment. One level of access may provide a client of the service with access to all of the request messages in the message schema for the service, and thus to substantially all of the functions provided by the service to clients in the distributed computing environment. Other levels may provide a client of the service with access to various subsets of the request messages in the message schema, and thus to various subsets of the functionality of the service. In one embodiment, levels of access may also be determined by a client's capabilities. For example a thin client may not be able to download large data files, and thus may be restricted from using a message requesting the download of a large data file.

In one embodiment, the client may provide information about the client to the authentication service to determine an access level for the client. In one embodiment, the information may include a request for a specific level of access to the service. In one embodiment, the gate factory may provide the information to the authentication service to determine the access level of the client. Thus, the gate factory may generate a client message gate that is capable of sending a subset of the entire set of messages described in the message schema to the service based upon the capabilities and/or access level of the client.

In step 1014, the gate factory has generated the client message gate, and may notify the client that the gate has been generated. In one embodiment, the client message gate is a distinct code module that is accessible by the client. In one embodiment, the client message gate resides on the client. The client may then generate messages and pass the messages to the client message gate, which may verify the messages and send the messages to the service. Embodiments of a gate pair mechanism for the client and service to exchange messages is further described in Figures 42a-42c. Embodiments of gate factories are further described elsewhere herein.

A gate comprises code and data, and thus may itself be passed in a message. This provides an alternative method for creating gates on clients and/or services. In one embodiment, a gate passed in a message may be cloned if a new client wishes to use the gate. The cloning process performs a new set of gate creation security checks including authentication of the new client. A new, unique authentication credential may be generated for the new client and embedded in the cloned gate.

Figure 42a - A client sending a message to a service

Figure 42a is a flow diagram illustrating a client sending a first message to a service according to one embodiment. In step 1020, the client may send a message to the client message gate. In one embodiment, the message may be an XML message. In step 1022, the message gate may embed an authentication credential in the message prior to sending the message. In one embodiment, the authentication credential may have been provided by an authentication service as part of gate construction as described above.

In one embodiment, the message gate may verify the data representation language type correctness, syntax, etc. of the message. In one embodiment, the message gate may compare the message to a message template in a data representation language message schema to determine data representation language type correctness of the message. In one embodiment, the message may be an XML message, and the message gate may check the message against an XML message schema. In one embodiment, the message schema may have

As described above, the client message gate may have embedded an authentication credential in the first message sent to the service. In step 1032, the service may send the authentication credential to an authentication service. In one embodiment, the authentication service may be the same authentication service used by the client to generate the authentication credential. In one embodiment, the service message gate may send the authentication credential to the authentication credential. In one embodiment, the entire message may be sent to the authentication service.

In step 1034, the authentication service may perform verification of the authentication credential. In one embodiment, the authentication service may include a copy of the authentication credential from the creation of the authentication credential. In one embodiment, the authentication service may compare the authentication credential received from the service with the copy of the authentication credential. If the authentication credentials match, in step 1036, the authentication service may notify the service that the authentication credential has been verified and appears to be valid. If the verification process fails, the authentication service may notify the service that the authentication credential appears to be invalid.

In one embodiment, the authentication service may establish an access level for the client to access the functionality of the service. In one embodiment, the client may have established an access level for the service with the authentication service. In one embodiment, the authentication service may notify the service of the access level of the client. The access level of the client may be used by the service to determine a subset of request messages as described in the service message schema that the client may send to the service.

In step 1038, if the authentication service notified the service that the authentication credential is valid in step 1036, the service may generate a service message gate to pair with the client gate to form a gate pair. The service message gate may include the authentication credential to embed in messages sent from the service to the client, and for comparison with the authentication credential in messages received from the client. The service message gate may also include an address (such as a UUID or URI) for the client message gate. The service message gate may also include access level information for the client for verifying that messages received from the client are in the subset of allowed messages the client may send to the service. The service message gate may also include a message schema for type checking and verifying the syntax of messages received from the client and for use in verifying if messages are in the allowed subset of messages. In one embodiment, the service may create a new service message gate. In another embodiment, a service message gate may already exist prior to step 1038 that may be used to generate the service message gate to communicate with the client. In this embodiment, the service may not create a new gate, but instead may update the existing gate with information about the message channel to be established between the client and the service.

In one embodiment, after the service generates the service message gate, the service may send a message to the client. The message may include information to identify the service message gate to the client message gate and thus to establish the communications channel between the client and the service using the message gate pair. In one embodiment, the message may include an address (such as a UUID or URI) for the service message gate.

Figure 42c - Exchanging messages with embedded authentication credentials

Figure 42c is a flow diagram illustrating the general process of a client and service exchanging messages with embedded authentication credential according to one embodiment. In one embodiment, after the client

In some embodiments, some services may not require authentication credentials for at least some clients. In one embodiment, a client wishing to access a service for which no authentication credential is required for the client may generate a message gate without using an authentication service. In another embodiment, an authentication service may return a null, empty or otherwise generic authentication credential to a client that does not require authentication to use a service. In one embodiment not requiring authentication, the message gates may send messages without embedding authentication credentials. In another embodiment, a null, empty or otherwise generic authentication credential may be embedded in messages by the message gates.

In one embodiment, the receiver message gate may verify the data representation language type correctness, syntax, etc. of the message upon receiving the message. In one embodiment, the receiver message gate may compare the message to a message template in a message schema to determine type correctness of the message. For example, the message may be an XML message, and the message gate may include an XML message schema. The receiver message gate may locate a message template for the message in the schema and compare the various XML items or fields in the message to the message template to determine type correctness of the items.

In one embodiment, the receiver message gate may check the allowability of the message. In one embodiment, the message may be a request message received from a client, and the message gate may determine if the requested function(s) specified by the message are in the subset of functions provided to the client by the access level the client established with the service through the authentication service. In one embodiment, the receiver message gate may compare the message to a subset of allowed request messages in a message schema to determine if the message is allowed.

In one embodiment, the sender and the receiver may verify the message for type correctness and/or allowability. In another embodiment, the sender may perform message verification. In yet another embodiment, the sender may not perform message verification, and the receiver may perform message verification. In still yet another embodiment, no verification may be performed.

Some clients may be too "thin" to support the full functionality of a client message gate. These clients may not perform some or all of the request message verification prior to sending request messages and the response message verification subsequent to receiving response messages as described above. For example, some simple client devices may include a small set of request messages that may be sent to a service, and a small set of responses that may be accepted from the service. In one embodiment, a minimal client message gate may be constructed for the client device that sends request messages and receives response messages without performing the message verification as described above. In another embodiment, a proxy client message gate may be set up on another device that may provide some or all of the message verification, sending, and receiving as described above for the client.

Figure 43 - Checking the integrity of messages

Figure 43 is a flow diagram illustrating a mechanism for checking the integrity of messages according to one embodiment. In step 1050, the sender gate, acting on behalf of a client or a service, may embed a token in a message to be sent. This token is separate and distinct from the authentication credential as described above. The token may include information allowing the receiving gate to verify that the message has not been compromised or

A bridging mechanism may be provided for "wrapping" one or more specific device discovery protocols, such as Bluetooth's, in a messaging API for the distributed computing environment. Wrapping may include framing the device discovery protocol with code and/or data (the API) so that the protocol can be run by clients and/or services in the distributed computing environment that would not otherwise be able to run it. When run, the bridging mechanism may allow for a discovery agent that discovers devices by a specific device discovery protocol to publish services for those devices in a space in the distributed computing environment. The services present an XML message schema interface to clients in the distributed network environment, and are capable of operating the various devices discovered by the specific device discovery protocol. Thus, service advertisements may be published for the services that operate the various devices discovered by the underlying wrapped device discovery protocols. The advertised services thus bridge devices (or services) external to the distributed network environment to clients on the distributed network environment.

Figure 27 illustrates one embodiment of a distributed computing environment with a space 1200. One or more discovery agents 1204 may participate in an external discovery protocol and bridge to the distributed computing environment through bridging mechanism 1202. When the wrapped device discovery protocols are run, discovery agents 1204 through bridging mechanism 1202 may publish service advertisements 1206a-1206c in space 1200, wherein each one of advertisements 1206a-1206c corresponds to a device or service discovered by one of discovery protocols 1204 outside the distributed computing environment. Clients may then access the external devices using the service advertisements 1206a-1206c in space 1200 to instantiate services on one of the agents 1204 that operates the corresponding external device or service.

Thus, clients of the distributed computing environment may use discovery agents wrapping device discovery protocols to find devices. A service acting as a bridge to these devices may be published in a space and advertised, so clients of the distributed computing environment may access the services provided by the external devices. The advertised service is a service within the distributed computing environment that is able to invoke a device outside the distributed computing environment via another protocol or environment, thus bridging the outside device/service to the distributed computing environment. A client within the distributed computing environment "sees" only the advertised service within the distributed computing environment and may not even be aware of the outside device/service.

In one embodiment, the distributed computing environment may provide a version of a space discovery message protocol, such as the discovery protocol described in the Spaces section, that may be mapped to an underlying external device discovery protocol, including the wrapped device discovery protocols described above. The mapped discovery protocol may register itself or be registered with a space, e.g. a default space, by placing an advertisement in that space. For each advertised discovery protocol, a subsequent results space to hold the results of the discovery protocol may be provided.

Figure 28 illustrates an example of the space discovery protocol mapped to a Bluetooth discovery service 1220 according to one embodiment. The Bluetooth discovery service 1220 may first register 1230 with the distributed computing environment. The Bluetooth discovery service 1220 may be wrapped in a bridging API, and an advertisement 1225 for the discovery service 1220 may be added 1232 in space 1224. A client or service may locate the discovery service advertisement 1225 on space 1224. When the discovery service 1220 is executed (utilizing the API wrapper as a bridge between the discovery protocol 1220 and the distributed computing

the agent may receive an event notifying of the removal of the Jini service. The agent may then remove the XML advertisement for the service from the space.

In one embodiment, to invoke a Jini service via an XML advertisement in a distributed computing environment space, a client may look up the service advertisement in the space and may send valid messages to the agent to access the service. The agent may invoke the proxy service corresponding to the Jini service by invoking the corresponding method through an RMI call to a service proxy. If the proxy is not instantiated, the agent may download the proxy code and instantiate a new instance of the proxy object. In one environment, every client connection may have a different proxy-instance. The incoming message from the client may be converted by the agent into a method call for the proxy. The result from the method call may be returned to the client as an outgoing message.

In one embodiment, only simple Java types may be used as arguments to an RMI method. If complex Java types are required, then one or more data advertisements may be passed as arguments to the call, where the data advertisements may indicate the location and access method of data for the complex Java types. In one embodiment, the agent may perform initial conversion from XML messages to an RMI method call invocation dynamically. Since, the agent knows the service interface, it may generate the corresponding set of messages that are advertised to the client.

Figure 29 illustrates bridging a client 1250 external to the distributed computing environment to a space 1254 in the distributed computing environment. Bridging agent 1252 may serve as the go-between between client 1250 and space 1254. Bridging agent 1252 may communicate with client 1250 in a communications protocol understandable by the client 1250. Bridging agent 1252 may map the client's communications protocol into the XML messaging protocol necessary to communicate with space 1254 perform the facilities provided by space 1254. Bridging agent 1252, at client 1250's request, may locate and run services on space 1254. For example, client 1250 may request a list of all services of a particular type from space 1254. Bridging agent 1252 may locate service advertisements 1256a-c and return the results to client 1250. Alternatively, the results may be posted in a results space, and the location of the results may be returned to the client 1250. Client 1250 may then choose to execute service advertisement 1256a, and may send a message (in the client 1250's communications protocol) to bridging agent 1252. Bridging agent 1252 may then send the XML request message(s) necessary to execute the service represented by service advertisement 1256a, and may return the results of the service to client 1250. Methods of handling the results of the service other than directly returning the results to the client 1250 may be used as described above in the section titled Spaces. Bridging agent 1252 thus may act as a service of the external client 1250 (via the external client's protocol) and as a client within the distributed computing environment to bridge a service within the distributed computing environment to the external client.

Sometimes, even within the distributed computing environment, clients and services cannot directly communicate with each other, only to a common space. In this case, the space service will automatically create a service proxy that bridges client to service. The proxy's main job is to route messages between client and service through the space. The service proxy may be created dynamically. The creation mechanism may be dependent upon space implementation. Refer to Figure 30 for an illustration of a proxy mechanism. A client 554 and a service 556 may not be able to communicate directly within the distributed computing environment, e.g., because they support different transport or network protocols. However, they both may be able to communicate with a

The bridge service or other agent may publish an advertisement for the bridge service (and thus for the enterprise service) in a space in the distributed computing environment. For example, a bridge service or other bridge agent may use Java Reflection to examine Beans for services in an enterprise system implemented with EJB, and then create service advertisements for bridge services to the Beans and publish those advertisements in spaces in the distributed computing environment. Reflection is a method for Java code to discover information about the fields, methods and constructors of classes, and to use reflected fields, methods, and constructors to operate on their underlying counterparts on objects, within security restrictions. The Reflection API accommodates applications that need access to either the public members of a target object or the members declared by a given class. Once the bridge services are advertised, clients may access the bridge services (and thus the corresponding enterprise services) similarly to any other advertised services in the distributed network environment, without knowledge of the architecture of the enterprise service providing the services.

Client Displays

There are several methods in which results from a service run by a client may be displayed in a distributed computing environment. Devices that may display results may include, but are not limited to: CRTs on computers; LCDs on laptops, notebooks displays, etc; printers; speakers; and any other device capable of displaying results of the service in visual, audio, or other perceptible format. The methods for displaying results may include, but are not limited to:

- The service may return results to a client directly or by reference, and the client may handle the display of the results.
- The service may return results to a client directly or by reference, and the client may pass the results to a display service directly or by reference, and the display service may display the results.
- The service may directly handle the displaying of the results.
- The service may pass the results to a display service directly or by reference, and the display service may display the results.

In the last method of displaying results, the client may specify the display service. For example, there may be a display service on or associated with the device on which the client resides that the client wishes to use to display the results of the service. When the client runs the service, the client may send a message to the service specifying the service advertisement of the client's display service. The service may then build a gate that allows it to send messages to the client's display service. Thus, when displaying results, the service invoked by the client becomes a client of the client's display service and sends its results (directly or by reference) for display to that display service. More detail on the client-service relationship, gates, and messaging is included in other sections of this document.

Conventional application models are typically based on predetermined, largely static user interface and/or data characteristics. Changes to conventional applications may require code modification and recompilation. The mechanisms described for advertising services and for specifying XML message schemas for communicating with services in the distributed computing environment may be used to provide a mechanism for applications (clients, services, etc) to describe dynamic display objects. Using the dynamic display objects, application behavior may

services acting on behalf of the application) may then access the schemas from the service advertisements to display data based upon formatting, data type, and other information stored in the schemas.

The following is an example of a schema containing dynamic display objects:

```
<element name="delivery" type="Space:shipto" minOccurs="0" />
<type name="TextField">
<element name="Address" type="string"/>
<element name="City" type="string"/>
<element name="State" type="string"/>
...
...
...
</type>
```

The above schema may be changed to the following without requiring an application recompile:

```
<element name="delivery" type="Space:shipto" minOccurs="0" />
<type name="TextField">
 <element name="Name" type="string"/>
 <element name="Address" type="string"/>
 <element name="City" type="string"/>
 <element name="State" type="string"/>
...
...
...
</type>
```

Figures 32A and 32B illustrate examples of using schemas of dynamic display objects according to one embodiment. In Figure 32A, application 1320 (may be a client, a service, or other application) has been implemented with presentation schema advertisement 1324 stored in space 1326. A presentation schema advertisement may include elements describing the data types, formatting specifications, fonts, locations, colors, and any other information used for displaying data for the application on display 1322. There may be multiple presentation schema advertisements for application 1320. For example, there may be one schema for each display page in a series of display pages (for example, Web pages on a Web site).

In one embodiment, application 1320 may invoke a discovery and/or lookup service to locate presentation schema advertisements. The discovery and/or lookup service may return an XML document listing one or more advertisements, and URIs to each of the schemas describing a particular display format, etc. Application 1320 may then select a presentation schema or schemas from the XML document. Application 1320 may then parse the schema, breaking out the elements of the schema into user interface components. The components then may be used to locate, format, and display results data on the appropriate display. The result data may be from running a service or from a results space, for example. Thus, as opposed to having a static or predetermined display, the application 1320 is configured to display results according to a presentation schema that may be dynamically changed without requiring a rebuild of the application.

An example of a string operation in C is the strlen() function, typically provided with standard C library implementations. The strlen() function takes a string pointer as input and returns the length (in bytes) of the string, not including the terminating character. For example, passing the character pointer 1450 to the strlen() function would return the length 12. The strlen() function may be implemented by "walking" the string until the terminating character is located, counting each character.

String copying in C is typically handled by a strcpy() or a strncpy() C library functions, which are implemented as:

```
char *strcpy(char *dest, const char *src);
char *strncpy(char *dest, const char *src, size_t n);
```

The strcpy() function copies the string pointed to by the character pointer src (including the terminating character) to the string pointed to by character pointer dest . The strings may not overlap, and the destination string dest must be large enough to receive the copy.

The strncpy() function is similar, except that not more than n bytes of src are copied. Thus, if there is no terminating character among the first n bytes of src, the result will not be terminated. If desired, an instruction may be placed in the code following a strncpy() to add a termination character to the end of the dest string. In the case where the length of src is less than that of n, the remainder of dest will be padded with nulls. The strcpy() and strncpy() functions return a pointer to the destination string dest .

Figure 33B illustrates an example of the results of the strncpy() function on string 1452, when strncpy() is called with the following parameters:

```
strncpy(string2, string1+3, 5);
```

where string2 is character pointer 1454 pointing to the first byte after the terminating character of string 1452, string1+3 is character pointer 1450 incremented by 3 bytes, and 5 is the number of characters (bytes) to be copied from the source location string1+3 to string2. After copying, the next character after the five characters copied from string1 may be set to the terminating character (the character may have been initialized to the terminating character prior to the copy). Thus, the two strings now occupy (13 + 6) = 19 bytes of memory. If the strcpy() function was applied with character pointer 1450 as the source string, the original string 1452 and the resultant new string would occupy (13 * 2) = 26 bytes.

Figure 33C illustrates an efficient method for representing and managing strings in general, and in small footprint systems such as embedded systems in particular. String 1452 is stored in memory as 12 bytes (no terminating character is required). String structure 1460 includes pointers (Address(A) and Address(L)) to the first and last characters of string 1452. Using this string structure, the string's length may be efficiently computed by subtracting the pointer to the first character from the pointer to the last character.

Operations such as string copy operations strcpy() and strncpy() may also be handled more efficiently. With string structures such as those illustrated in Figure 33C, a new string structure 1462 may be created, and the first and last character pointers may be initialized to point to the respective characters in string 1452. Thus, a portion or all the string 1452 does not have to be copied to new storage for the string. As strings can be hundreds or even thousands of characters long, the memory saved using the string structures and string methods implemented to take advantage of them may be considerable. This method of handling copies of portions or all of

Using the string structures with the recursive processing allows the processing to be done without creating copies of the subsections for processing. Copying of subsections may be particularly costly in recursive processing, because as the recursion goes deeper, more and more copies of the same data are made. Using the string structures, only the string structure containing the pointers to the first and last bytes in the subsection needs to be created and passed down to the next level. Other operations, such as determining the length of a subsection, may be performed efficiently using the address information stored in the string structures. Also, by using the string structures, terminating characters such as those used to terminate C strings are not necessary, conserving memory in small footprint devices such as embedded devices.

XML representation of Objects

As previously mentioned, Jini RMI may not be practical for some clients, such as thin clients with minimal memory footprints and minimal bandwidth. The serialization associated with the Jini RMI is slow, big, requires the JVM reflection API, and is a Java specific object representation. Java deserialization is also slow, big and requires a serialized-object parser. Even Java based thin clients may not be able to accept huge Java objects (along with needed classes) being returned (necessarily) across the network to the client, as required in Jini.

A more scalable distributed computing mechanism may be provided by embodiments of a distributed computing environment. A distributed computing environment may include an API layer for facilitating distributed computing. The API layer provides send message and receive message capabilities between clients and services. This messaging API may provide an interface for simple messages in a representation data or meta-data format, such as in the eXtensible Mark-up Language (XML). Note that while embodiments are described herein employing XML, other meta-data type languages or formats may be used in alternate embodiments. In some embodiments, the API layer may also provide an interface for messages to communicate between objects or to pass objects, such as Java objects. Objects accessible through API layer 102 are represented by a representation data format, such as XML. Thus, an XML representation of an object may be manipulated, as opposed to the object itself.

The API layer may sit on top of a messaging layer. The messaging layer may be based on a representation data format, such as XML. In one embodiment, XML messages are generated by the messaging layer according to calls to the API layer. The messaging layer may provide defined static messages that may be sent between clients and services. Messaging layer may also provide for dynamically generated messages. In one embodiment, an object, such as a Java object, may be dynamically converted (compiled) into an XML representation. The object may include code and/or data portions. The object's code and/or data portions may be compiled into code and data segments identified by XML tags in the XML representation. The messaging layer may then send the XML object representation as a message. Conversely, the messaging layer may receive an XML representation of an object. The object may then be reconstituted (decompiled) from that message. The reconstitution may examine the XML representation for tags identifying code and/or data segments of the XML representation, and use information stored in the tags to identify and decompile the code and/or data portions of the object.

may be used on both the client and/or the service in the compilation and decompilation processes. In one embodiment, XML schema(s) to be used in the compilation and decompilation of Java objects may be passed by the service to the client in the service advertisement.

XML provides a language- and platform-independent object representation format. Thus, the process as illustrated in Figure 34 where an object is compiled into an XML representation of the object and decompiled to reproduce the object may not be limited to moving Java objects, but in some embodiments may be applied to moving objects of other types between entities in a network.

Figure 46 illustrates a mechanism for sending objects from services to clients using XML representations of the objects. In step 2200, a client may request an object from a service. In one embodiment, the object may be a Java object. In one embodiment, the client may send a message (e.g. an XML message) to the service requesting the object. In step 2202, the service, after receiving the request, may provide the object to a compilation process. In one embodiment, the compilation process may be integrated in the virtual machine within which the service is executing. In one embodiment, the virtual machine may be a Java Virtual Machine (JVM). In one embodiment, the compilation process may be provided as an extension to the JVM. In step 2204, the compilation process generates a data representation language representation of the object. In one embodiment, the data representation language is XML. The representation of the object may include a name or identifier for the object and one or more instance variables for the object. An instance variable, in object-oriented programming, is one of the variables of a class template that may have a different value for each object of that class. In step 2206, the service may send the data representation language representation of the object to the client. In one embodiment, the representation may be packaged in a message. In one embodiment, the message is in the data representation language. Message gates, as described elsewhere herein, may be used for message passing between the client and service.

In step 2208, the client receives the data representation language representation of the object and provides the representation to a decompilation process. In one embodiment, the decompilation process may be integrated in the virtual machine within which the client is executing. In one embodiment, the virtual machine may be a Java Virtual Machine (JVM). In one embodiment, the decompilation process may be provided as an extension to the JVM. In step 2210, the decompilation may generate a copy of the object from the data representation language representation of the object. The object may then be provided to the client.

JVM compilation/decompilation API

Figures 35a and 35b are data flow diagrams illustrating embodiments where a virtual machine (e.g. JVM) includes extensions for compiling objects (e.g. Java Objects) into XML representations of the objects, and for decompiling XML representations of (Java) objects into (Java) objects. The JVM may supply an Applications Programming Interface (API) to the compilation/decompilation extensions. The client 1500 and service 1502 may be executing within JVMs. The JVMs may be on the same device or on different devices.

In both Figure 35a and Figure 35b, the JVM XML compiler/decompiler API 1530 may accept a Java object 1510 as input, and output an XML representation of the object 1510 and all its referenced objects (the object graph of object 1510) in an XML data stream 1514. In addition, the JVM XML compiler/decompiler API 1530 may accept an XML data stream 1522, which includes an XML representation of object 1520 and all its

serialization. Also, an object may be complied or decompiled by a single call to the JVM XML compiler/decompiler API.

In addition, integrating the compilation/decompilation of objects with the JVM may allow the compilation and decompilation of objects to be performed faster than methods using reflection and serialization because, in the object traversal model implemented with reflection and serialization, the code outside the JVM does not know the structure or graph of the Java object, and thus must traverse the object graph, pulling it apart, and ultimately must repeatedly call upon the JVM to do the compilation (and the reverse process for decompilation). This process may be slowed by the necessity of making repeated calls to the JVM, outside the code. Having the compilation and decompilation functionality integrated with the JVM, as described herein, avoids having to make repeated calls from code outside the JVM to the JVM. In one embodiment, an object may be complied or decompiled by a single call to the JVM XML compiler/decompiler API.

In one embodiment, the compilation/decompilation functionality may be implemented as a service in the distributed computing environment. The service may publish a service advertisement in a space. A process in the distributed computing environment may use a search or discovery service to locate the compilation/decompilation service. The process (a client of the service) may then use the service by passing Java objects to be compiled into XML representations and/or XML representations to be decompiled into Java objects to the service.

Java objects may include code (the object's methods) and data. An object's code may be non-transient; the code does not change once the object is created. An object's data, however, may be transient. Two objects created from the same Java class may include identical code, but the data in the two objects may be different. In one embodiment, the compilation function may compile a Java object's data into an XML representation of the object, but may not include the object's actual code in the XML representation. In one embodiment, information about the object may be included in the compiled XML representation to indicate to the receiver how to recreate the code for the object. The XML representation may then be stored in an XML data stream and sent (e.g. in a message) to a receiving process (client or service). The receiving process may then pass the XML data stream to the decompilation function. The decompilation function may then decompile the XML data stream to produce the Java object including its data. In one embodiment, the code for the object may be reproduced by the decompilation function using information about the object included in the XML representation, as the code for an object may be statically defined and the JVM receiving the object may be able to reproduce the code (if necessary) using its knowledge of the object.

In one embodiment, the XML representation of an object produced by the compilation function may include the Java object's data and information about the Java object. The information may include class information for the Java object. An object signature may be included in the information and may be used to identify the object's class, etc. The decompilation function may recreate the code for the Java object using the information about the Java object and may decompile the data from the XML data stream into the Java object. Thus, a complete object including its code and data may be reproduced on the JVM executing the receiving client or service from the decompiled data and the information describing the object. In one embodiment, the information describing the object may be stored in one or more XML tags. In one embodiment, the client or service receiving the XML data stream may include an XML schema that describes the object, and the XML schema may be used to reconstruct the Java object from the decompiled data and from the information about the

send, as a client of the store space service, the XML representations of the objects to the store space to be stored for possible later access, or for access by other processes.

Security issues in the Decompilation of XML Representations of Objects

Spaces, as described herein, may serve as a file system in the distributed computing environment. Security may be provided for files in the system in the form of access rights. Access rights may be checked each time a file is accessed (opened, read, or written to). Thus, a method for providing file access security in the distributed computing environment may be desirable. This method may also be applied to the XML representations of Java objects that may be stored in spaces and transmitted between clients and services in the distributed computing environment.

In one embodiment, a user of a client on a device in the distributed computing environment may be identified and authenticated when first accessing the client. In one embodiment, the user may supply a physical identification such as a smart card for identification and authorization. In another embodiment, a challenge-response mechanism (such as user ID and password) may be used for identification and authorization. Yet another embodiment may use electronic identification such as a digital signature for identification and authorization. Any other method of identification and authorization may be used.

Once identified and authorized, the user may then perform various operations on the client, including accessing one or more services in the distributed computing environment. During these operations, as described above, one or more objects may be created (locally) or acquired from elsewhere (e.g. from services or spaces). The objects may be modified and may be compiled into XML representations of the objects and stored locally by the client or sent to a space service for (transitive or persistent) store. Some of the objects may be received from services (store services or other services) in the form of XML representations of the objects, which may be decompiled by the XML compiler/decompiler to recreate the objects on the client.

In one embodiment, during the decompilation of the XML representation of objects, each XML message may be checked to verify that the user has access rights to the object. If the user does not have the proper access rights, the XML compiler/decompiler may not decompile the object for the user. In one embodiment, a security exception may be thrown by the XML compiler/decompiler. In one embodiment, the user may be informed of the access violation.

Access right information, such as the creator and access levels allowed (creator-only access, read only, read/write, delete, copy, etc.) for the object may be embedded in the XML message(s) containing the XML representation of the object. Access authorization may be determined during the identification and authorization of the user. For example, the object may allow "read only" access for most users, and "read/write" access for the creator of the object. If the user tries to access an object using read/write access rights, and the user did not create the object, the decompilation process may detect this as an access violation, and may disallow the access and notify the user.

In one embodiment, when the user is done using the client, the user may log off or otherwise signal the user is finished with the client (e.g. remove a smart card). Objects created on the client by decompilation may be automatically deleted when the client detects that the user is finished. This may prohibit future users from intentionally or accidentally accessing the user's objects. In one embodiment, all objects created by decompilation

representations of Java objects and/or objects implemented in other languages, etc. Existing object storage technologies tend to be language and/or operating system specific. These storage systems also tend to be too complicated to be used with small footprint systems such as embedded systems.

JavaSpaces in Jini is an existing object repository mechanism. A JavaSpace may be only capable of storing Java objects and may be too large to be implemented in small devices with limited amounts of memory. Each object in a JavaSpace may be serialized as previously described, and thus has the same limitations as previously described for the reflection and serialization techniques.

A store mechanism may be provided for the distributed computing environment that may be heterogeneous (not language or operating system dependent), that may scale from small to large devices, and that may provide transient or persistent storage of objects. In one embodiment, the store mechanism in the distributed computing environment may be implemented as an Internet Web page or set of pages defined in the XML markup language. XML provides a language- and platform-independent object representation format enabling Java and non-Java software to store and retrieve language-independent objects. Since the store mechanism is on the Web, devices of all types and sizes (small to large) may access the store mechanisms. Web browsers may be used to view the store mechanism implemented as Web pages. Web search engines may be used to search for contents in the store mechanism implemented as Web pages. Internet administration mechanisms (existing and future) and XML tools may be used to administer the XML-based store mechanisms.

In one embodiment, the store mechanisms may be used to store objects created, represented or encapsulated in XML. Examples of objects that may be stored in the store mechanisms may include, but are not limited to: XML schemas, XML representations of objects (for example, Java objects compiled into XML representations as described above), service advertisements, and service results (data) encapsulated in XML. In one embodiment, to prevent unauthorized access of an XML object, an authorization credential such as a digital signature or certificate may be included with the XML object, and a client wishing to access the XML object may be required to have the proper authorization credential to access the XML object. In one embodiment, the store mechanism may be a space as described in the Spaces section herein.

Store mechanisms may be services in the distributed computing environment. A store mechanism implemented as a service may be referred to as a "store service". A store service may publish an advertisement in a space. The space itself is an example of a store service. Some store services may be transient. For example, a space service that stores service advertisements may be a transient store. Other store services may be persistent. For example, a store service that stores results from services may be a persistent store.

Figure 36 illustrates a client 1604 and a service A 1606 accessing store mechanisms 1600 and 1602 in the distributed computing environment according to one embodiment. This illustration is intended to be exemplary and is not intended to be limiting to the scope of this invention. In one embodiment, store mechanisms 1600 and 1602 may each be an Internet Web page or set of Web pages defined in XML and accessible by a Web browser and other Internet tools. Store mechanism 1600 is a transient store capable of storing objects implemented using XML. Store mechanism 1602 is a persistent store also capable of storing objects implemented using XML. Service A 1606 may publish an XML service advertisement 1608 in transient store 1600. Persistent store may also publish an XML service advertisement in transient store 1600 (or on another transient store in the distributed computing environment). At some point, client 1604 may require functionality provided by Service A 1606.

file system in the distributed computing environment and may include access checks and other security mechanism as described herein.

Dynamically Converting a Java Object into an XML Document

In one embodiment, the distributed computing environment may provide a mechanism to convert and represent an object class instance into an XML document. In order to send representation of a class instance to another service, the object may be converted and represented as an XML document. In one embodiment, when receiving an XML document, a program may instantiate a class instance corresponding to the object represented by the document. In one embodiment, the objects may be Java objects, and the program may be a Java program.

In one embodiment, an intermediary format may be used to represent an XML document and may be dynamically processed to generate a class instance that represents the XML document. The class may define a set of instance variables and "set and get" methods to access the instance variables. A corresponding XML document may be defined as a set of tags, with one tag for each instance variable. When the document is parsed, a hashable representation may be constructed where the hash key may include the instance variable name and the value may include the instance variable value. If there are multiple occurrences of a complex instance variable, an enumeration value may be stored in a hash table. In one embodiment, the process may be limited to only one level of complex types for the instance variables, and the elements may be homogeneous.

In one embodiment, a protected instance variable may be added to the class definition that may include the name of the corresponding class. The XML document representation may use the class name as the document type. Having the class name embedded in the document may allow dynamic instantiation of the right class instance when the object is reconstructed.

In one embodiment, upon receiving a document, a class instance generator method may be used to extract the class type and to parse the document to generate the intermediary hash table representation. The generator method may instantiate a new class instance and may use the set methods to initialize the instance object from the hash table values. In one embodiment, since the class type is defined and the hash table is generic, this process may be performed for any class that matches the above class definition.

In one embodiment, the reverse process may also be performed where a class instance may be processed into the intermediary hash table representation and a generator method may be used to produce an XML document from the hash table representation. This process may also be made generic so that it may be performed for any XML document that matches the above specification.

This method is not intended to be limited to Java Class objects, and may be applied to other computer-based objects, including class object instances of other programming languages. In addition, the method is not intended to be limited to XML representations of object instances; other representation formats including other data representation languages (such as HTML) may be substituted for XML.

XML-Based Process Migration

The distributed computing environment may enable the distribution and management of distributed applications. For example, the distributed computing environment may include mobile clients that are dockable with stations that provide monitors, printers, keyboards, and various other input/output devices that are typically not provided on mobile devices such as PDAs, cell phones, etc. These mobile clients may run one or more

Figure 37 illustrates process migration using an XML representation of the state of a process according to one embodiment. Process A 1636a may be executing on node 1630. Process A 1636a may be a client or service. At some point during the execution of Process A 1636a, the state of execution of Process A 1636a may be captured and stored in an XML-encapsulated state of Process A 1638. The execution of Process A 1636a on node 1630 may then be stopped. Later, node 1632 may locate the XML-encapsulated state of Process A 1638 and use it to resume Process A 1636b on the node 1632. Resuming Process A may include using the stored state 1638 to resume thread execution, recalculate transient variables, re-establish leased resources, and perform any other functions necessary to resume execution as recorded in the stored XML state of the process 1638.

The following is an example of using XML-based process migration in the distributed computing environment, and is not intended to be limiting. A mobile client device may be connected to node 1630 and executing Process A 1636a. The user of the mobile client device may desire to stop execution of Process A 1636a on node 1630, and to resume execution at a later time at another (or the same) node. In one embodiment, the user may be prompted with a query to determine if the user wishes to store the state of Process A 1636a and resume execution later. If the user replies in the affirmative, the XML-encapsulated state of the process may be captured and stored in persistent store 1634. Later, the user may connect the mobile computing device at node 1632. In one embodiment, the user may then execute process 1636b and select a "Resume from Stored State" option. The node 1632 may then search for and locate the XML-encapsulated state of Process A 1638, download it, and use it to resume Process A 1636b. Alternatively, the process may itself know that it was "suspended" on another node, and may resume from the stored state without user input.

Figure 48 illustrates a mechanism for migrating processes using data representation language (such as XML) representations of the processes in a distributed computing environment. In step 2260, a process is executing within a first device. In step 2262, a current computation state of the process is converted into a data representation language representation of the current computation state, wherein the computation state of the process comprises information about the execution state of the process within the first device. This information may include, but is not limited to, information about one or more threads of the process, information about one or more leases to services held by the process, and information about one or more objects of the process (such as objects held by the threads), wherein an object is an instance of a class in a computer programming language (such as the Java programming language).

In step 2264, the data representation language representation of the current computation state of the process may be stored. In one embodiment, the representation may be stored locally. In one embodiment, the data representation language representation of the current computation state of the process may be stored to a space using a space service. In one embodiment, the data representation language representation of the current computation state of the process may be sent to the space service in one or more messages.

In step 2266, a second device may access the stored data representation language representation of the current computation state of the process. In one embodiment, an advertisement may be provided for the stored data representation language representation, and the second device may receive or access the advertisement and use information in the advertisement to locate and access the stored data representation language representation. In step 2268, the process may be reconstituted at the current computation state within the second device from the

include a GPS location or locations of the local distributed computing environment, entities represented by space services within the environment, and/or the various services advertised in the spaces in the environment.

In one embodiment, wired connections may be provided to a local distributed computing network. In this environment, a user with a mobile client device may "plug in" directly to the network using a wired connection "docking station". Examples of wired connections include, but are not limited to: Universal Serial Bus (USB), FireWire, and twisted-pair Internet. In one embodiment, a docking station may also provide input/output capabilities such as a keyboard, mouse, and display for the mobile client device. In this embodiment, the location of the mobile client device may be provided to the lookup or discovery mechanism by the docking station.

In one embodiment, a mobile client device may connect to a distributed computing network. As the user of the mobile client device navigates within wireless communications range of the distributed computing network, the mobile client device may constantly, or at various intervals, provide a location vector as input to the local lookup or discovery mechanism. The mobile client device may obtain the location vector from a GPS system built into or associated with the mobile client. In one embodiment, the client may send its location information (e.g. via XML messaging) to a local service discovery mechanism, such as one of the space location mechanisms described herein. For example, the client may run the space discovery protocol specifying discovery for spaces offering services within a certain range of the client's location, or the client may instantiate a space search service to search for spaces advertising services provided for the client's vicinity.

As the mobile client device moves into a specified range of a space within the distributed computing environment, the services and/or data stored in the space may be made available to the mobile client device. In embodiments where the client device regularly provides its location to a discovery mechanism, local services and/or data may automatically be made available to the client's user. In one embodiment, the user of the mobile client device may determine the specified range of a space. For example, the user may choose to display all restaurants within one mile of a current location. Alternatively, the range may be specified in the configuration of the local distributed computing network. For example, a citywide distributed computing network may be configured to provide its services to all users within three miles of the city limits. In one embodiment, visual indicators, for example icons, representing the various services and/or data offered by the space may be displayed on the mobile client device. The client may then access one or more of the displayed services and/or data. In one embodiment, information from two or more spaces may be displayed simultaneously on the mobile client device. In one embodiment, the user may select what services and/or data are to be detected. For example, in a shopping mall, a user with a mobile client device may choose to display all shoe stores in the mall.

In one embodiment, executable code and/or data used in the execution of the code may be downloaded to the mobile client device to allow the user to execute an application provided by a service in the space. For example, moviegoers with mobile client devices may download interactive movie reviews from services in a space for the movie theater, and may thus perform real-time feedback about the movie they are watching. In one embodiment, an XML object compilation/decompilation mechanism as described elsewhere herein may be used to compile the code and/or data to produce XML representations of the code and/or data, and to decompile the XML representations to reproduce the code and/or data on the mobile client device. In one embodiment, an executable version of a process may previously exist on the mobile client device, and a stored state of the process may be downloaded to the mobile client device to allow the user to execute the process using the stored state. In one

range of one of the spaces, the advertisements offered by that space may be removed from the mobile client device 1700's display.

In one embodiment, advertisements on a space may include location information for the services or data that they provide. Thus, discovery mechanism 1706 may determine when mobile client device 1700 moves within a specified range of a particular service advertised on space 1718, and may provide (or remove) the service advertisement based upon the location of the mobile client device 1700.

Computing devices are shrinking while at the same time gaining power and functionality. Storage devices, CPUs, RAM, I/O ASICS, power supplies, etc. have been reduced in size to where small, mobile client devices may include much of the functionality of a full-sized personal computer. However, some components of a computer system are not easily shrinkable because of the human factor and other factors. These components include, but are not limited to: keyboards, monitors, scanners, and printers. The limits on reducing the size of some components may prevent mobile client devices from truly assuming the role of personal computers.

In one embodiment, docking stations may be provided that allow users with mobile client devices to connect to and use components that are not available on the mobile client device because of human or other factors. For example, docking stations may be provided in public places such as airports or libraries. The docking stations may provide monitors, keyboards, printers or other devices for users with mobile client devices. In one embodiment, the docking stations may not fully function without help from a real computing device such as a mobile client device connected by a user. The docking station may provide services such as various input/output functions to the client using the computing power of the mobile client device.

A docking station may provide one or more connection options to a mobile client device. The connection options may include wireless connections and wired connections. Examples of wireless connections include, but are not limited to: infrared such as IrDA and wireless network connections similar to those provided by a network interface card (NIC) in a notebook computer. Examples of wired connections include, but are not limited to: USB, FireWire, and twisted-pair Ethernet.

A mobile client device may discover the location of docking stations using a method substantially similar to that described above for mobile client devices. The location of one or more docking stations in a local distributed computing network may be discovered using a discovery mechanism to discover spaces with advertisements for docking stations. The mobile client device may provide a location to the discovery mechanism. In one embodiment, the discovery mechanism or a lookup mechanism may return the location of one or more docking stations closest to the location of the mobile client device. Alternatively, the discovery mechanism or lookup mechanism may return a URI of the space containing the advertisements for the docking stations, and the mobile client device may then connect with the space to provide the location of the one or more docking stations near the device. In one embodiment, the mobile client device may supply information to the lookup or discovery mechanism to specify requirements such as monitor resolution, screen size, graphics capabilities, available devices such as printers and scanners, etc. In one embodiment, information about the one or more docking stations may be supplied to the user on the mobile client device including availability (is another user using the docking station), components and capabilities of the various docking stations.

When a user approaches a docking station, a claiming protocol may be initiated. When the docking station accepts the claim, secure input and output connections may be established between the mobile client device

In one embodiment, a user may connect a mobile client device to a docking station without using the discovery mechanism. For example, a user in an airport may visually detect a docking station and connect a mobile client device to it. Another example may be a library providing a docking station room with a plurality of docking stations for use, where users may access any of the docking stations that are available.

5

9. The method as recited in claim 8, wherein the information representing the computer programming language method call further includes one or more parameter values of the method call, and wherein said executing a computer programming language method in accordance with the regenerated method call comprises providing the one or more parameter values from the information representing the method call as parameter values of the method call.

10. The method as recited in claim 8, wherein the service comprises a service method gate configured to provide an interface to computer programming language methods of the service by receiving messages and invoking methods specified by the messages, and wherein said regenerating the method call is performed by the service method gate.

11. The method as recited in claim 1, wherein said performing a function generates results data, the method further comprising the service providing the generated results data to the client.

12. The method as recited in claim 1, wherein said performing a function generates results data, the method further comprising:

storing the generated results data to a space service in the distributed computing environment;

providing an advertisement for the stored results data to the client, wherein the advertisement comprises information to enable access by the client to the stored results data; and

the client accessing the stored results data from the space service in accordance with the information in the provided advertisement.

13. The method as recited in claim 12, wherein the client accessing the stored results data comprises:

generating a client results message endpoint in accordance with the information in the provided advertisement, wherein the client results message endpoint is configured to send messages to the space service for the client;

generating a results request message, wherein the results request message requests a portion of the results data to be provided to the client;

the client results message endpoint sending the results request message to the space service; and

the space service sending the requested portion of the results data to the client results message endpoint in response to receiving the results request message.

14. The method as recited in claim 13, wherein the results request message is received on the space service by a space service results message endpoint, and wherein the space service sending the requested portion of the results to the client results message endpoint comprises:

generating a results response message, wherein the results response message includes the requested portion of the results; and

the space service results message endpoint sending the results response message to the client results message endpoint.

23. The system as recited in claim 17, wherein, in said performing a function, the service device is further configured to execute a computer programming language method in accordance with the information representing the computer programming language method call included in the message.

24. The system as recited in claim 17, wherein, in said performing a function, the service device is further configured to:

regenerate the computer programming language method call in accordance with an identifier of the method call included in the message; and

execute a computer programming language method in accordance with the regenerated method call.

25. The system as recited in claim 24, wherein, in said executing a computer programming language method, the service device is further configured to provide one or more parameter values included in the message as parameter values of the method call.

26. The system as recited in claim 17, wherein said performing the function generates results data, and wherein the service device is further configured to provide the generated results data to the client device.

27. The system as recited in claim 17, further comprising:

a space service device configured to receive and store results data from service devices in the distributed computing system;

wherein said performing the function generates results data, and wherein the service device is further configured to:

store the results data to the space service device; and

provide an advertisement for the stored results data to the client device, wherein the advertisement comprises information to enable access by the client device to the stored results data.

28. The system as recited in claim 27, wherein the client device is further configured to access the stored results data from the space service device in accordance with the information in the provided advertisement for the stored results data.

29. The system as recited in claim 28, wherein, in said accessing the stored results data, the client device is further configured to:

generate a client results message endpoint in accordance with the information in the provided advertisement, wherein the client results message endpoint is executable within the client device, wherein the client results message endpoint is configured to:

generate a results request message, wherein the results request message requests a portion of the results data be provided to the client device; and

send the results request message to the space service device.

37. The device as recited in claim 33, wherein the service is further operable to store results data generated by the function to a space service in the distributed computing environment, and wherein the client component is further configured to:

access an advertisement for the results data, wherein the advertisement comprises information to enable access by the client component to the results data; and

access the results data from the space service in accordance with the information in the provided advertisement for the stored results data.

38. The device as recited in claim 33, wherein said computer programming language is the Java programming language, and wherein the information representing a method call in the message represents a Java method call to a Java method implemented on the service.

39. A device comprising:

a client component configured to generate a message, wherein the message includes information representing a computer programming language method call; and

a message endpoint configured to send the message to a service in a distributed computing environment;

wherein the service is operable to perform a function on behalf of the client component in accordance with the information representing the computer programming language method call included in the message.

40. The device as recited in claim 39, wherein the service is further operable to perform the function on behalf of the client component asynchronously to processing of the client component.

41. The device as recited in claim 39, wherein the client component is further configured to generate the computer programming language method call, and wherein said generating a message is performed in response to said generating the computer programming language method call.

42. The device as recited in claim 39, wherein the device further comprises a virtual machine executable within the device, wherein the client component and the message endpoint are executable within the virtual machine.

43. The device as recited in claim 42, wherein the virtual machine is a Java Virtual Machine (JVM).

44. The device as recited in claim 39, wherein the service is further operable to store results data generated by the function to a space service in the distributed computing environment, and wherein the client component is further configured to:

access an advertisement for the results data, wherein the advertisement comprises information to enable access by the client component to the results data; and

51. A carrier medium comprising program instructions, wherein the program instructions are computer-executable to implement:

a client generating a message, wherein the message includes information representing a computer programming language method call;

the client sending the message to a service, wherein the service is configured to perform functions on behalf of the client; and

the service performing a function on behalf of the client in accordance with the information representing the computer programming language method call included in the message.

52. The carrier medium as recited in claim 51, wherein the service device performs the function on behalf of the client device asynchronously to processing on the client device.

53. The carrier medium as recited in claim 51, wherein the client comprises a client method gate, wherein, in said generating a message, the program instructions are further computer-executable to implement:

the client method gate receiving the computer programming language method call from a process executing within the client; and

the client method gate generating the message for the client.

54. The carrier medium as recited in claim 53, wherein the process is executing within a virtual machine, wherein the virtual machine is executing within a client device in the distributed computing environment.

55. The carrier medium as recited in claim 54, wherein the virtual machine is a Java Virtual Machine (JVM).

56. The carrier medium as recited in claim 51, wherein the information representing the computer programming language method call includes an identifier of the method call, and wherein, in said performing a function, the program instructions are further computer-executable to implement:

regenerating the method call in accordance with the identifier of the method call included in the information representing the method call; and

executing a computer programming language method in accordance with the regenerated method call.

57. The carrier medium as recited in claim 56, wherein the information representing the computer programming language method call further includes one or more parameter values of the method call, and wherein, in said executing a computer programming language method in accordance with the regenerated method call, the program instructions are further computer-executable to implement providing the one or more parameter values from the information representing the method call as parameter values of the method call.

58. The carrier medium as recited in claim 51, wherein the program instructions are further computer-executable to implement:

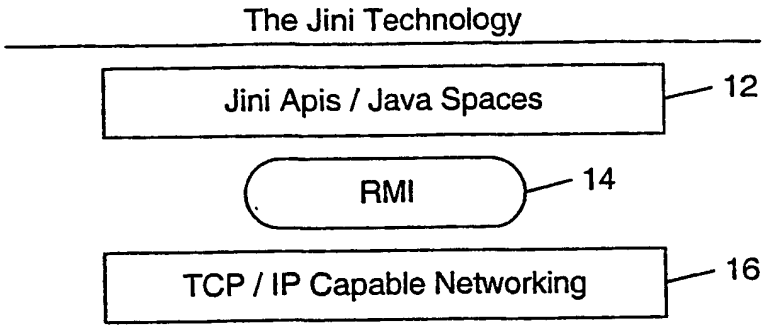


Fig. 1
(Prior Art)

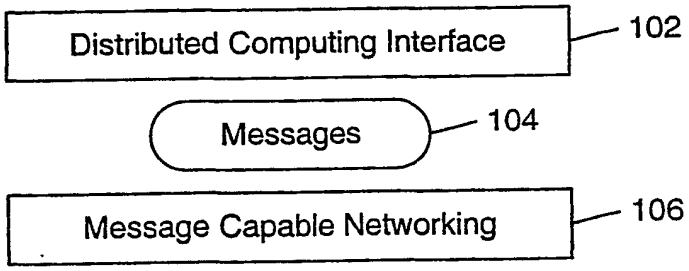


Fig. 2

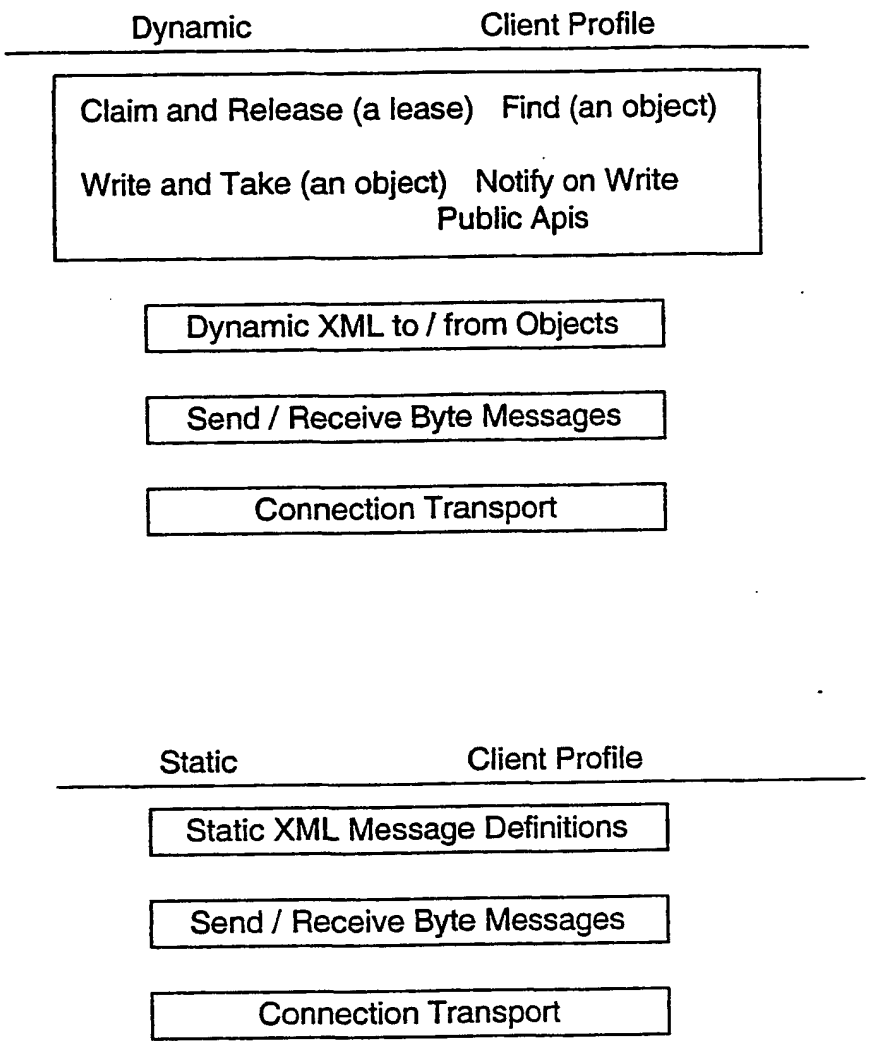


Fig. 5

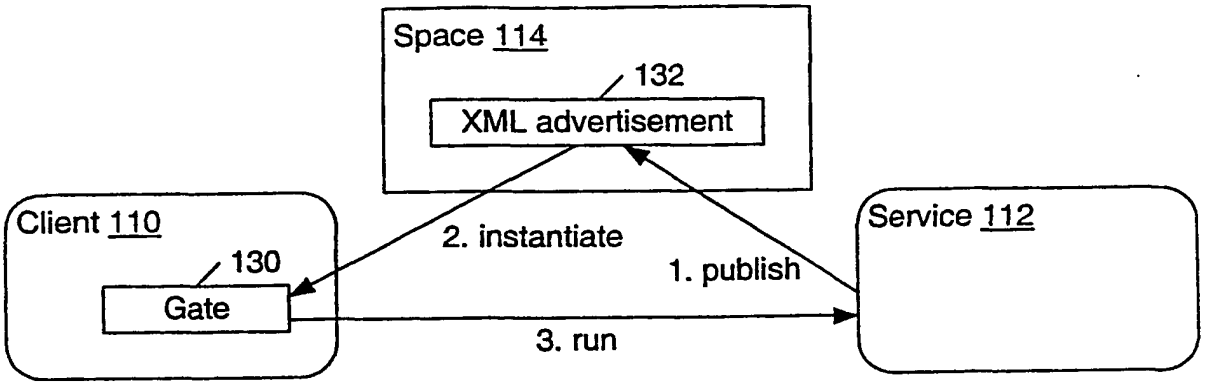


Fig. 8

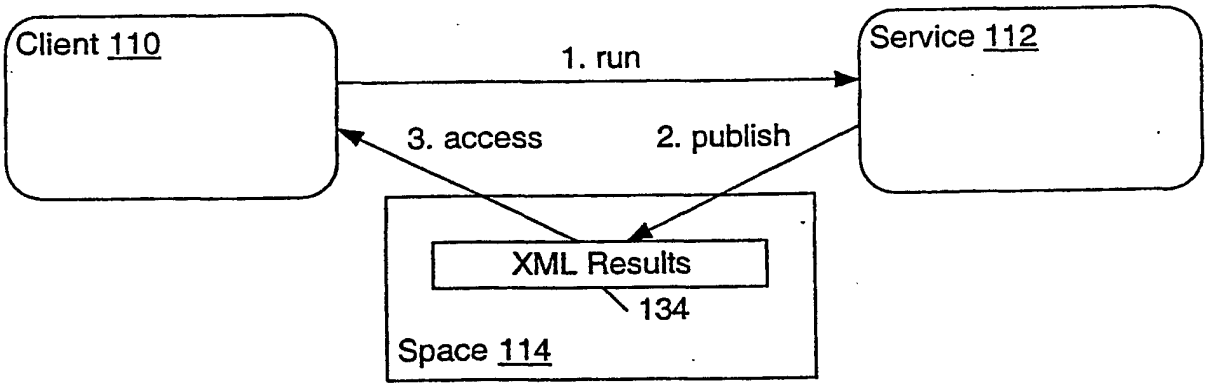


Fig. 9

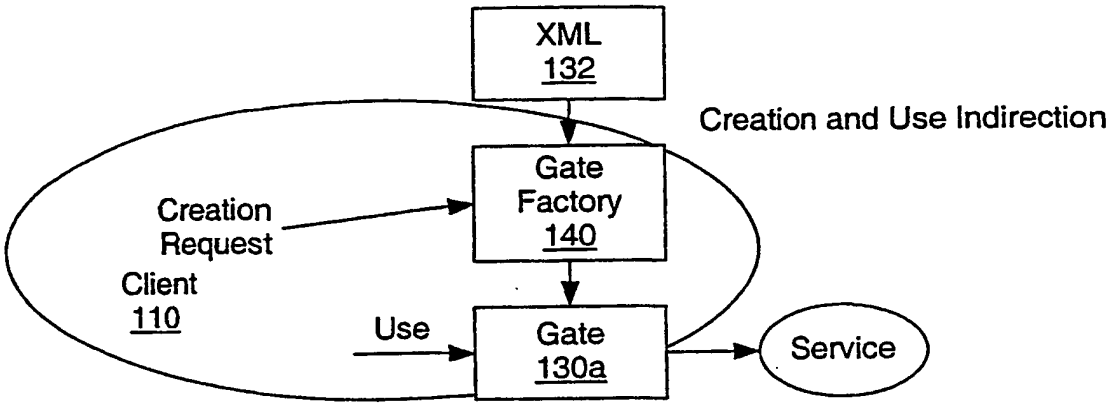


Fig. 11a

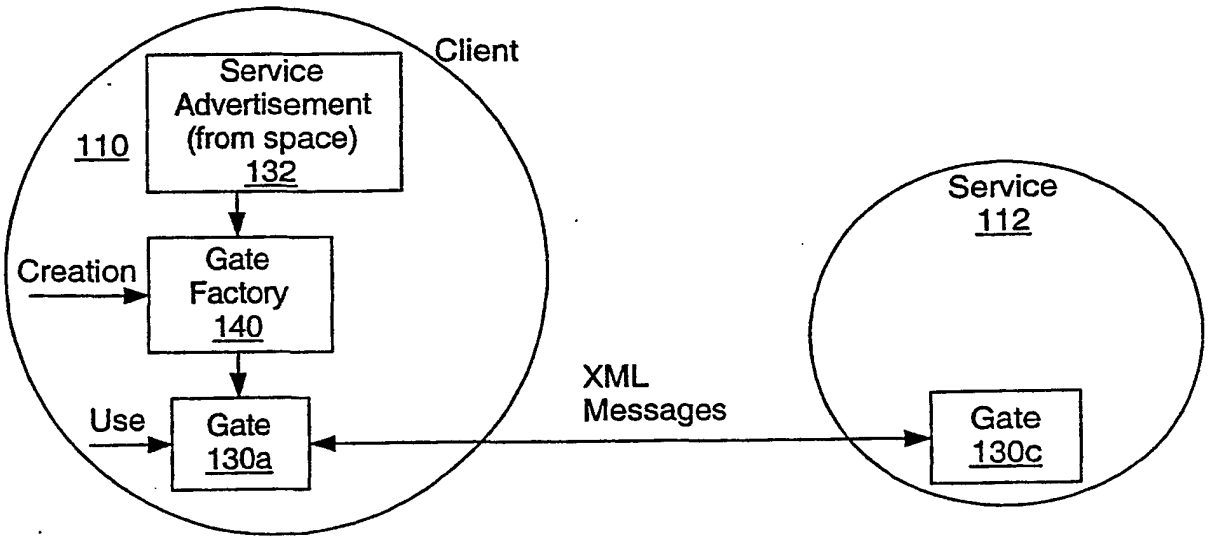


Fig. 11b

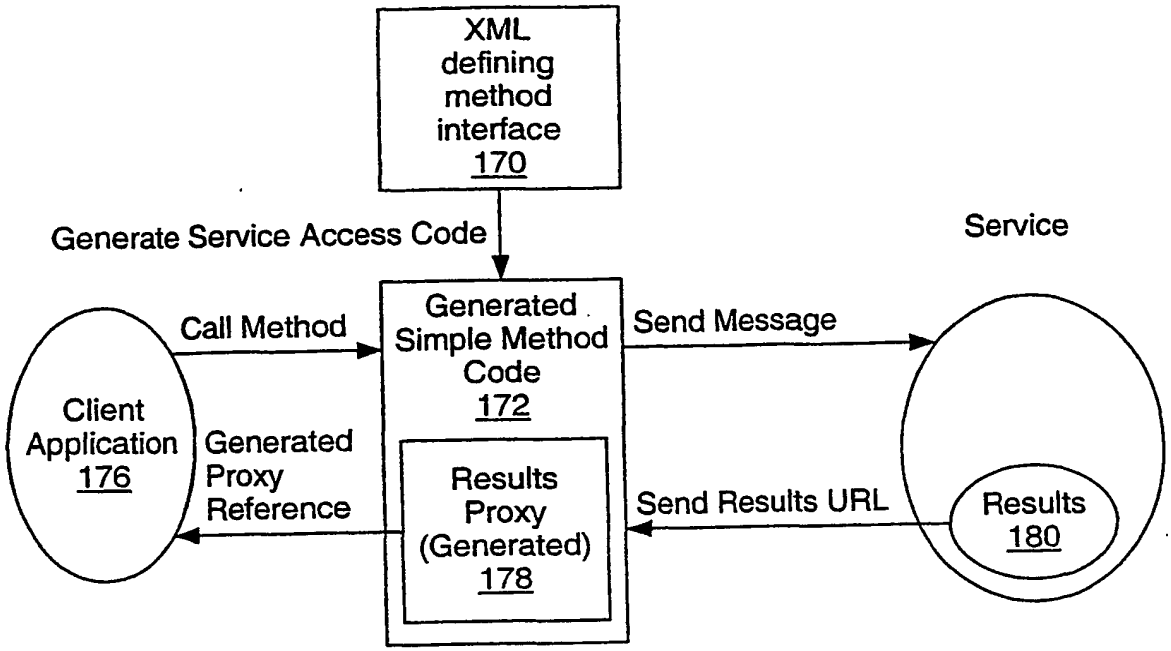


Fig. 14

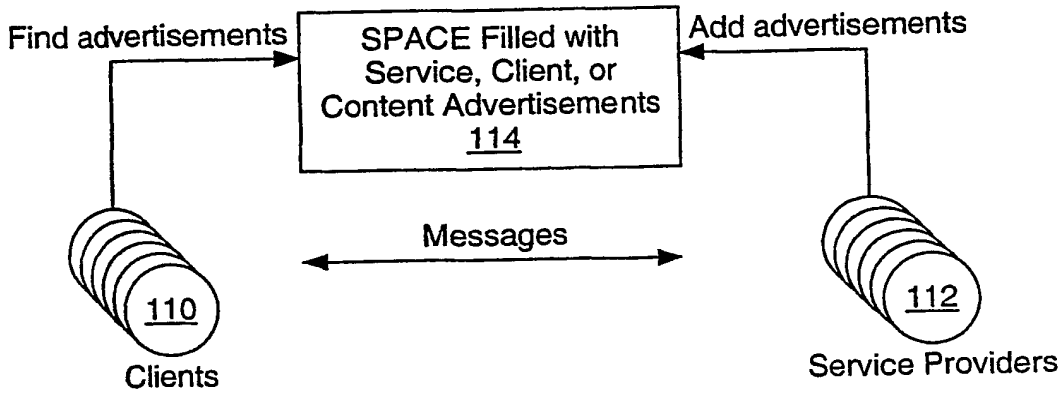


Fig. 15

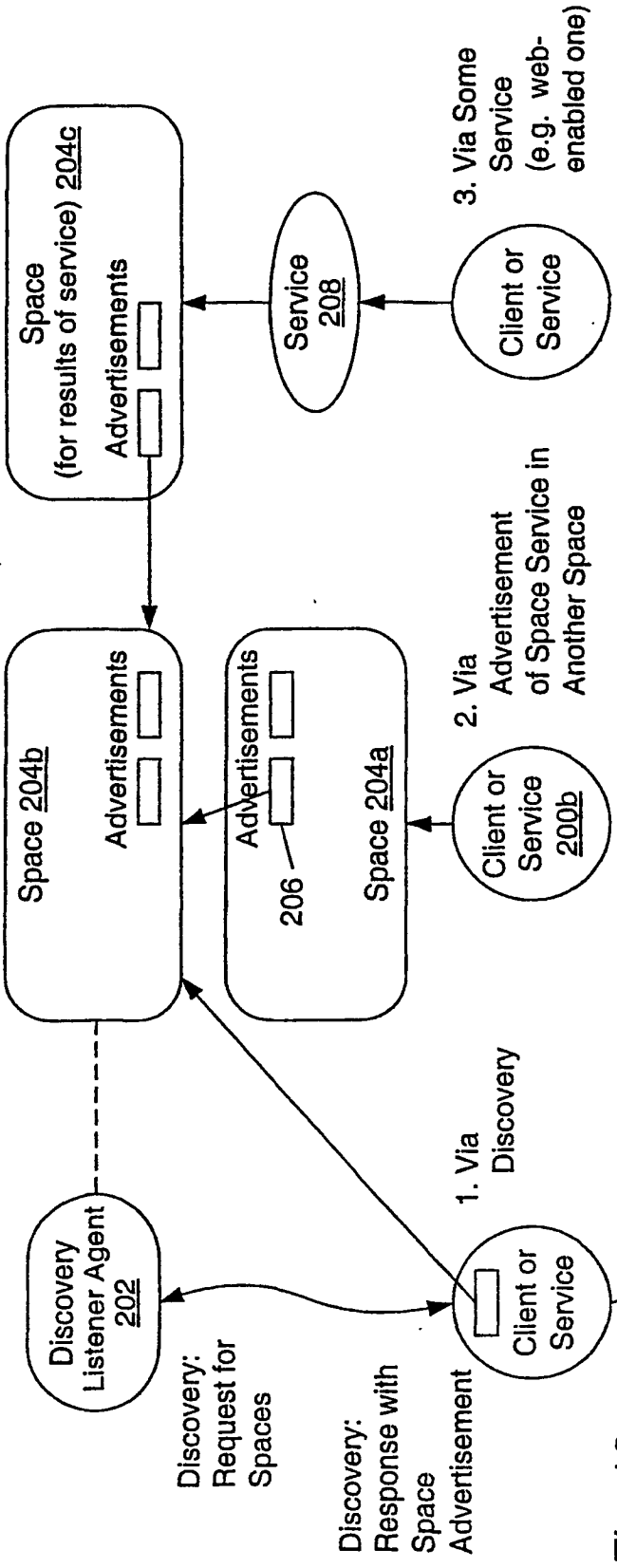


Fig. 18

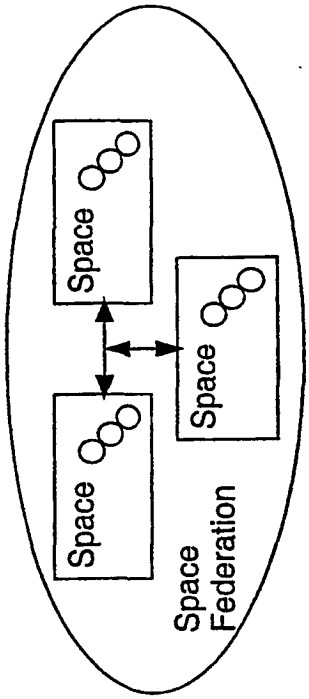


Fig. 19

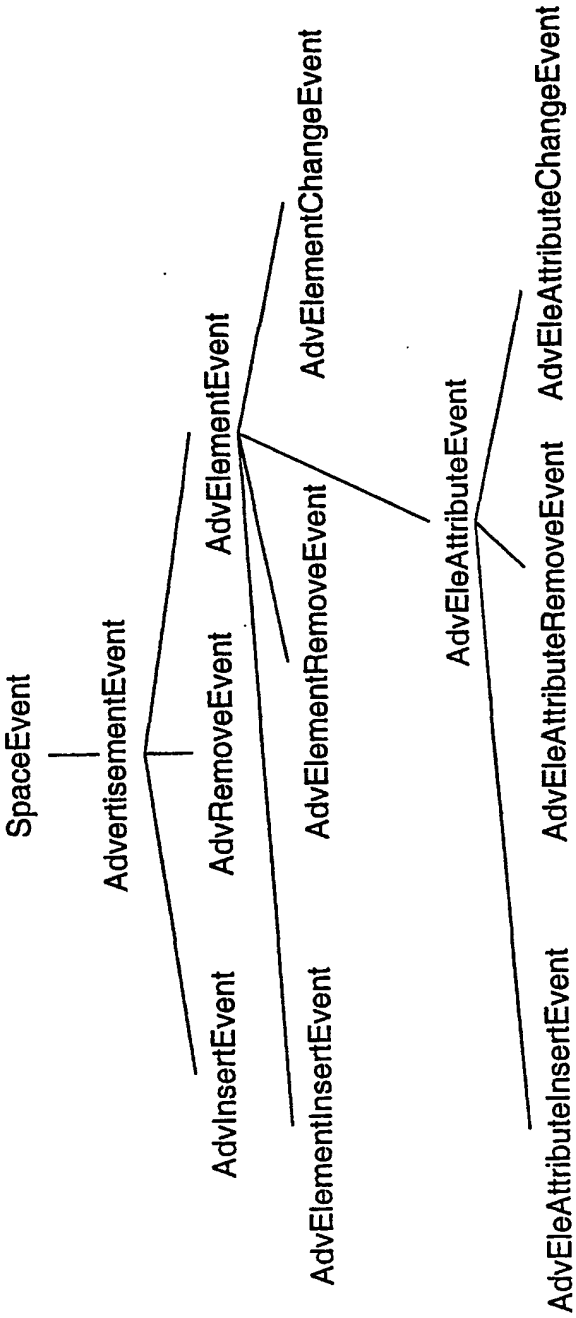


Fig. 21

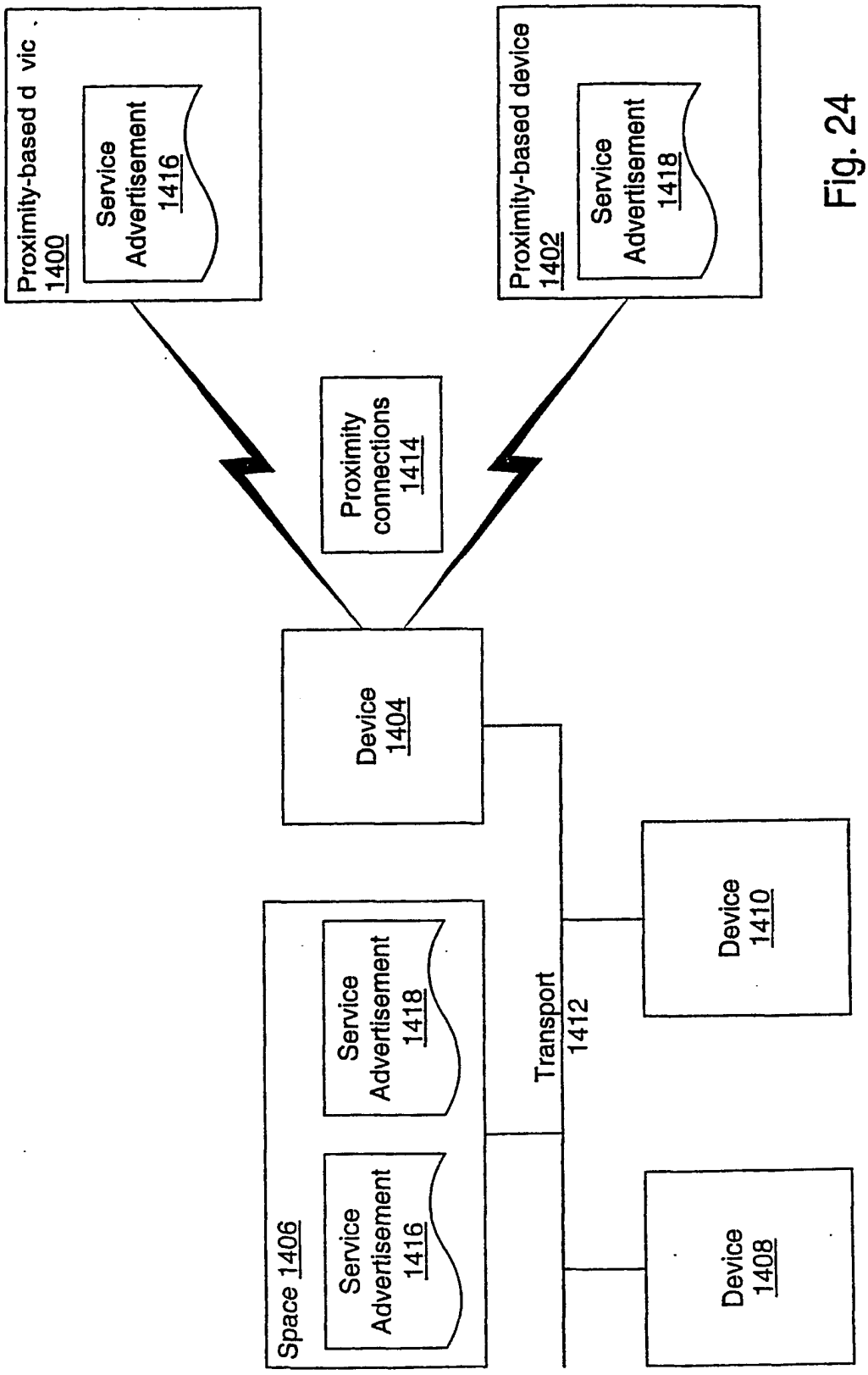


Fig. 24

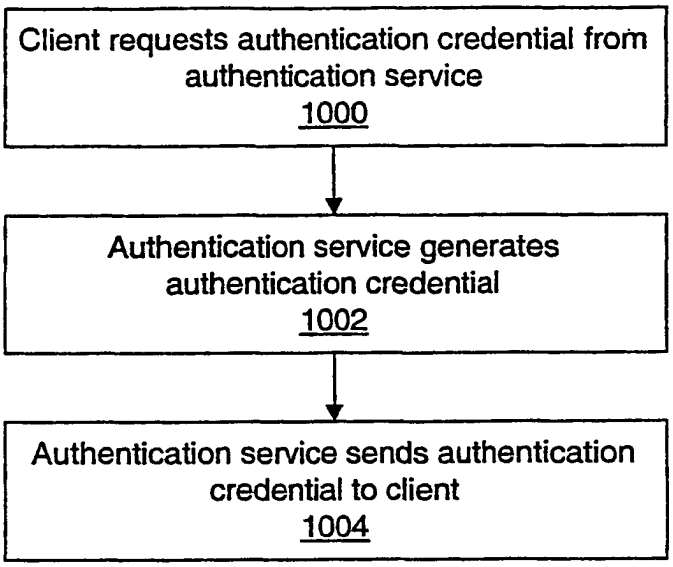


FIG. 26a

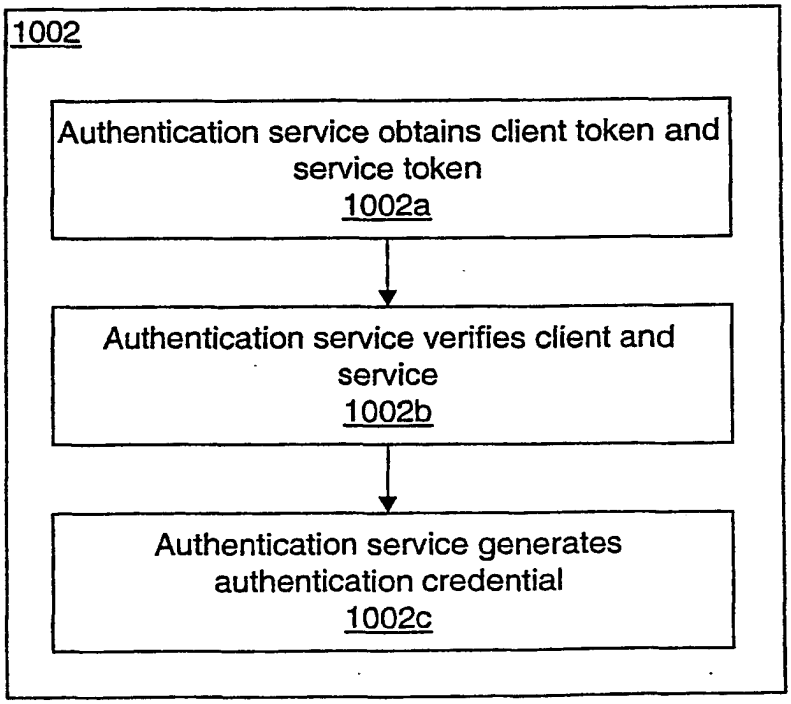


FIG. 26b

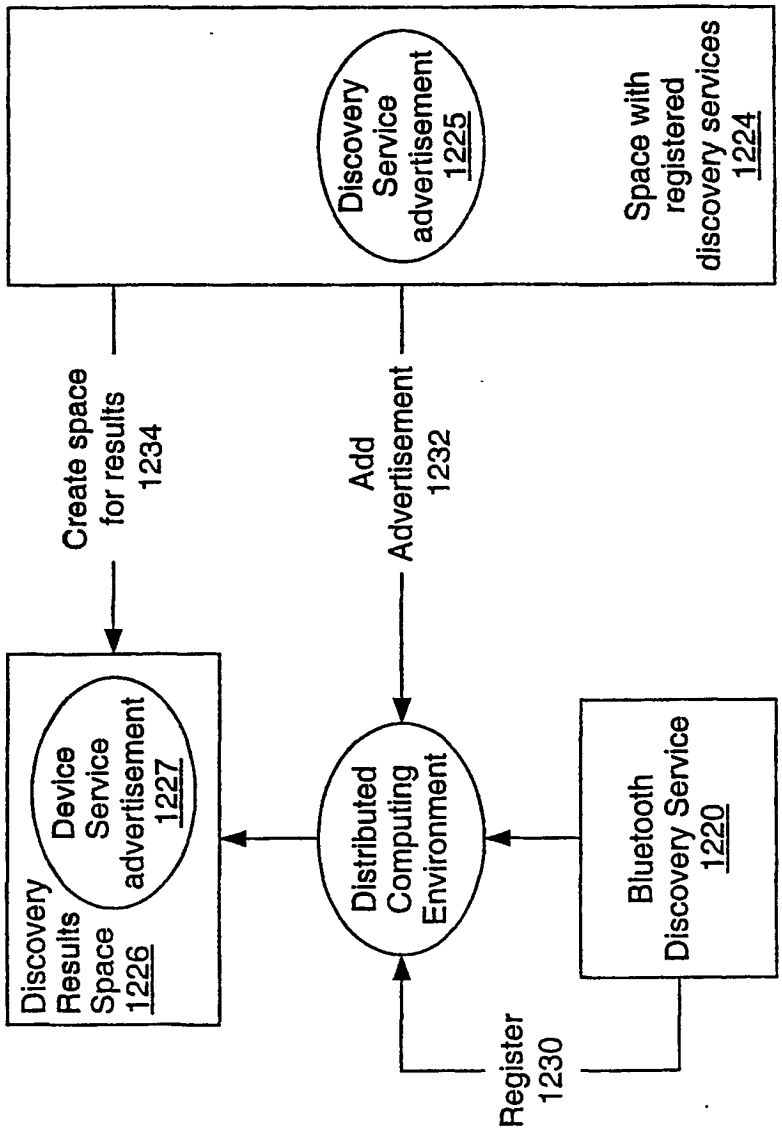
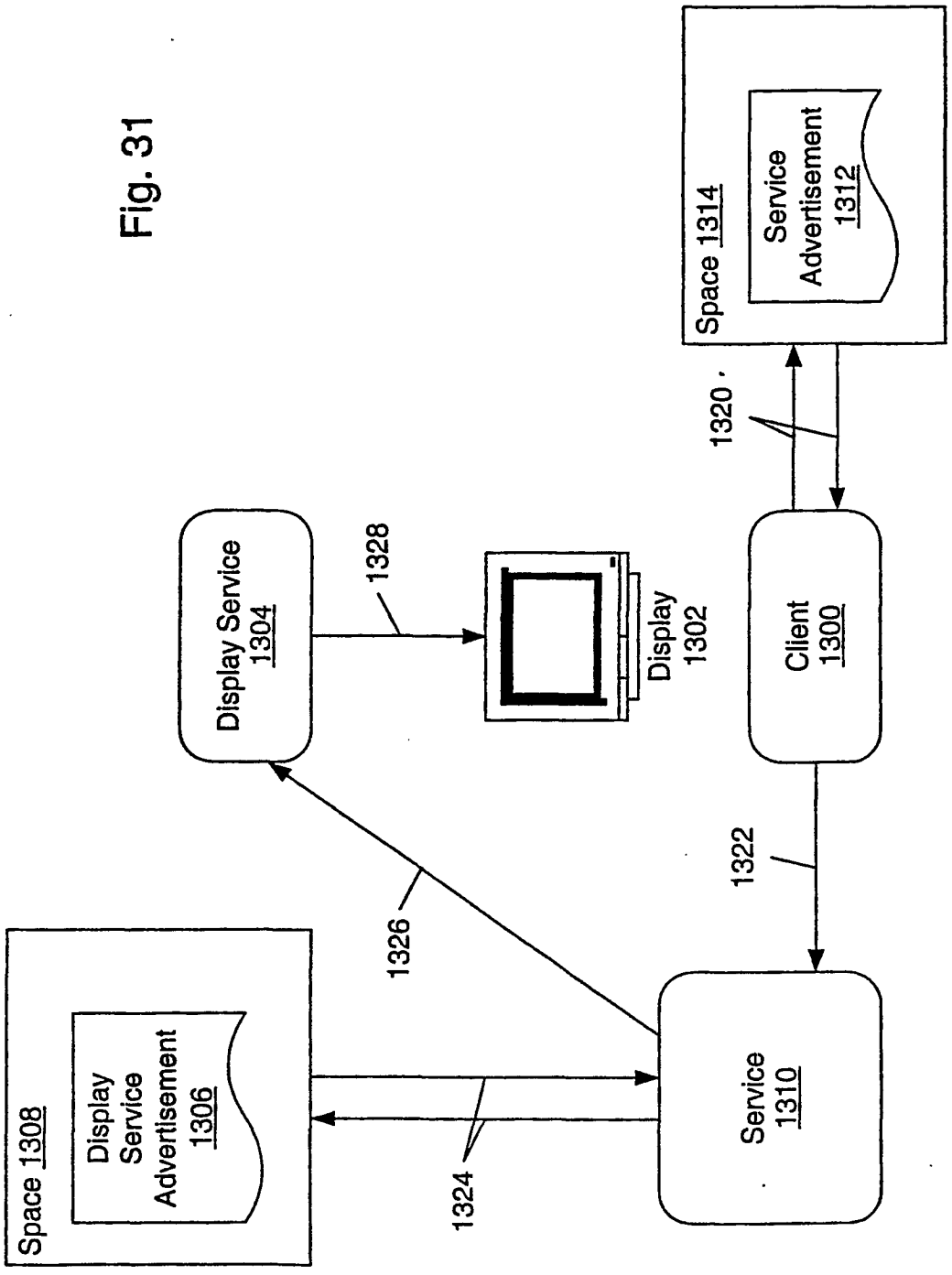


Fig. 28

Fig. 31



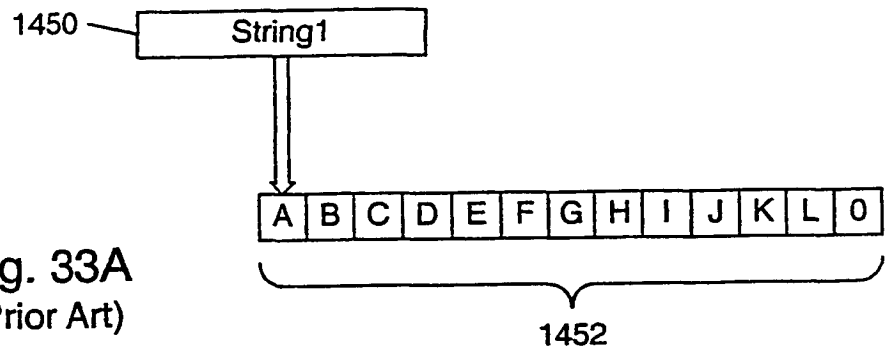


Fig. 33A
(Prior Art)

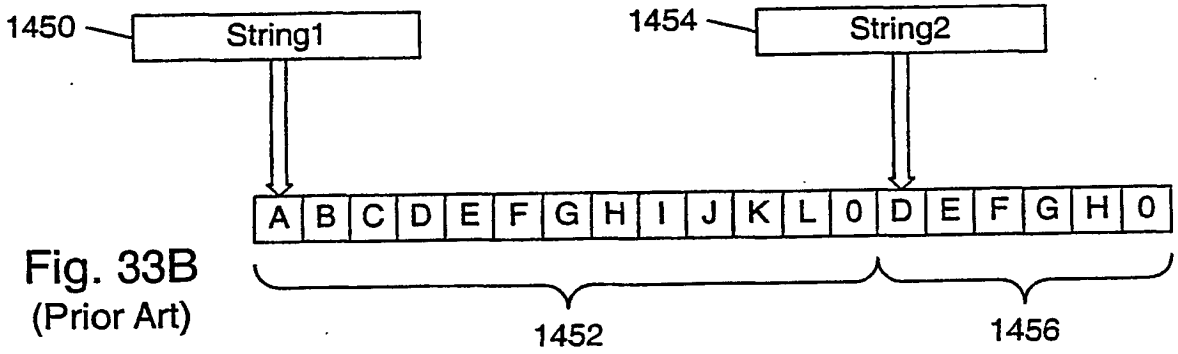


Fig. 33B
(Prior Art)

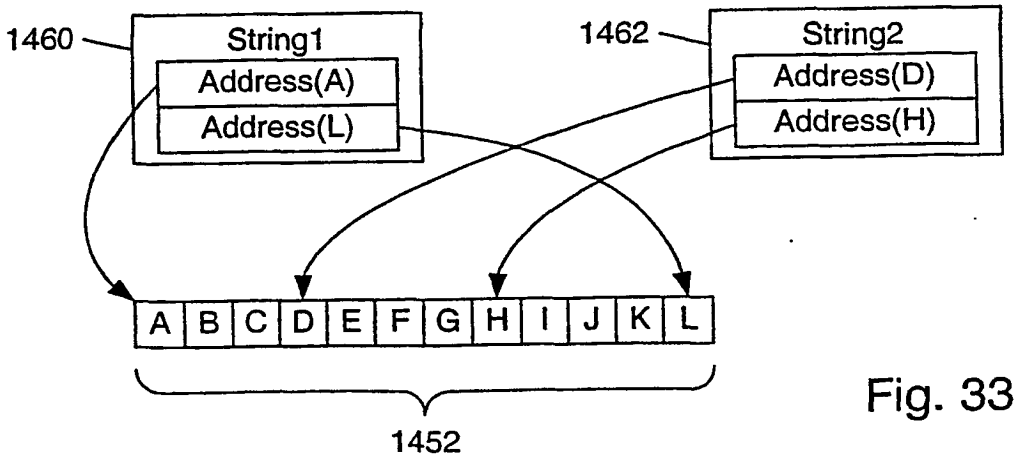


Fig. 33C

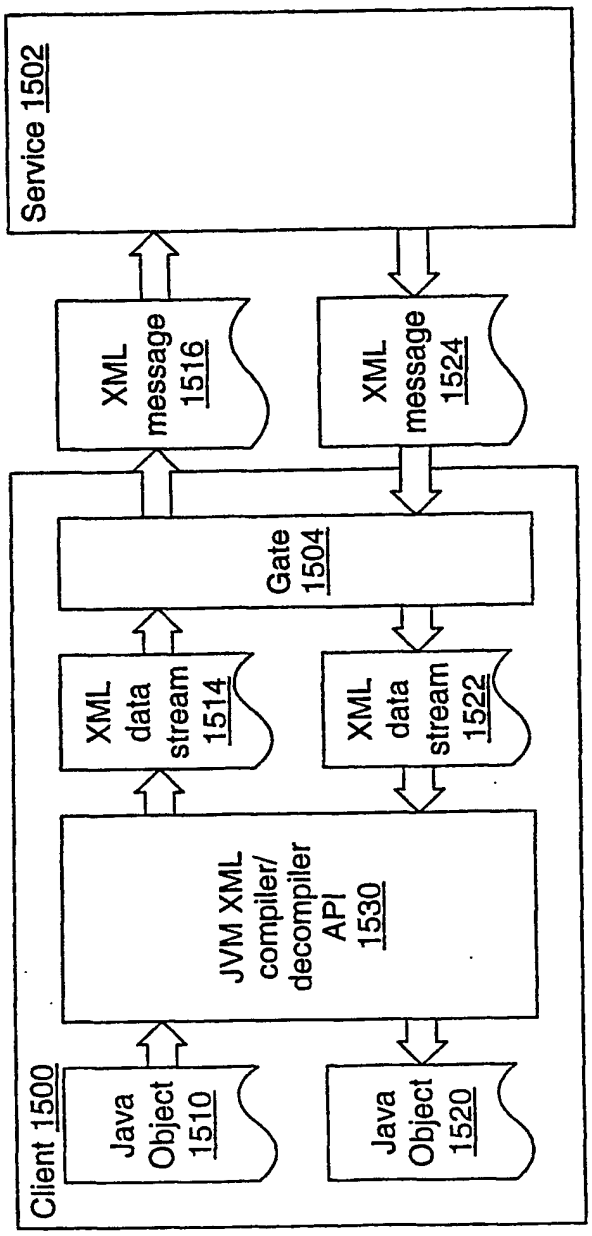


Fig. 35a

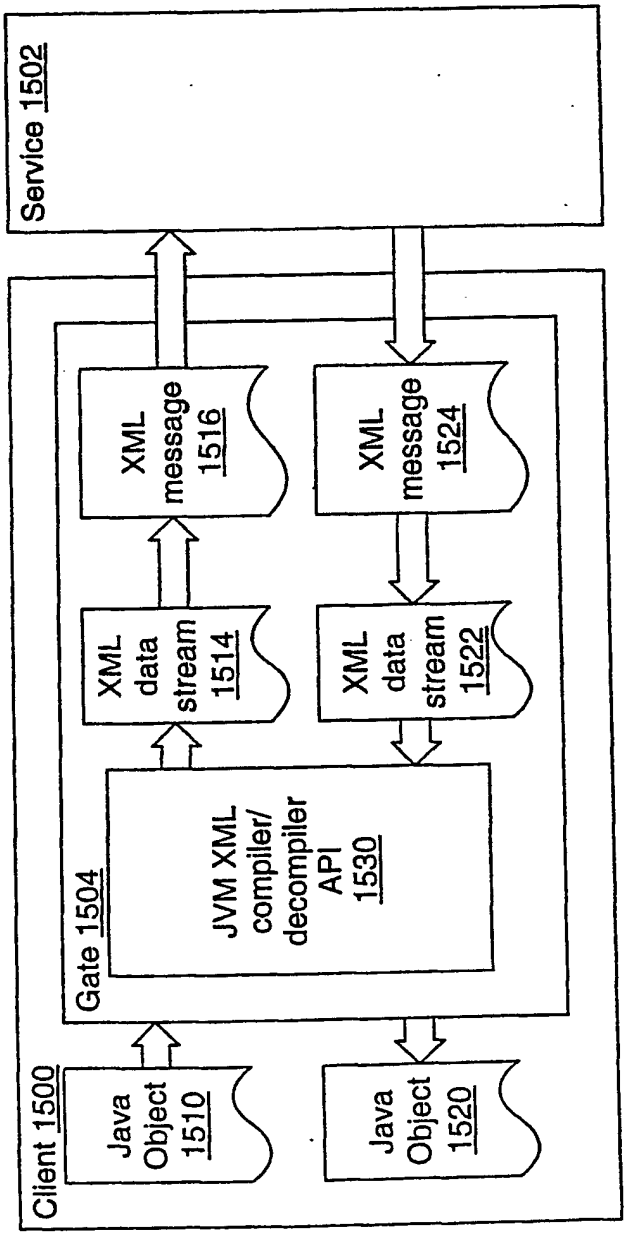


Fig. 35b

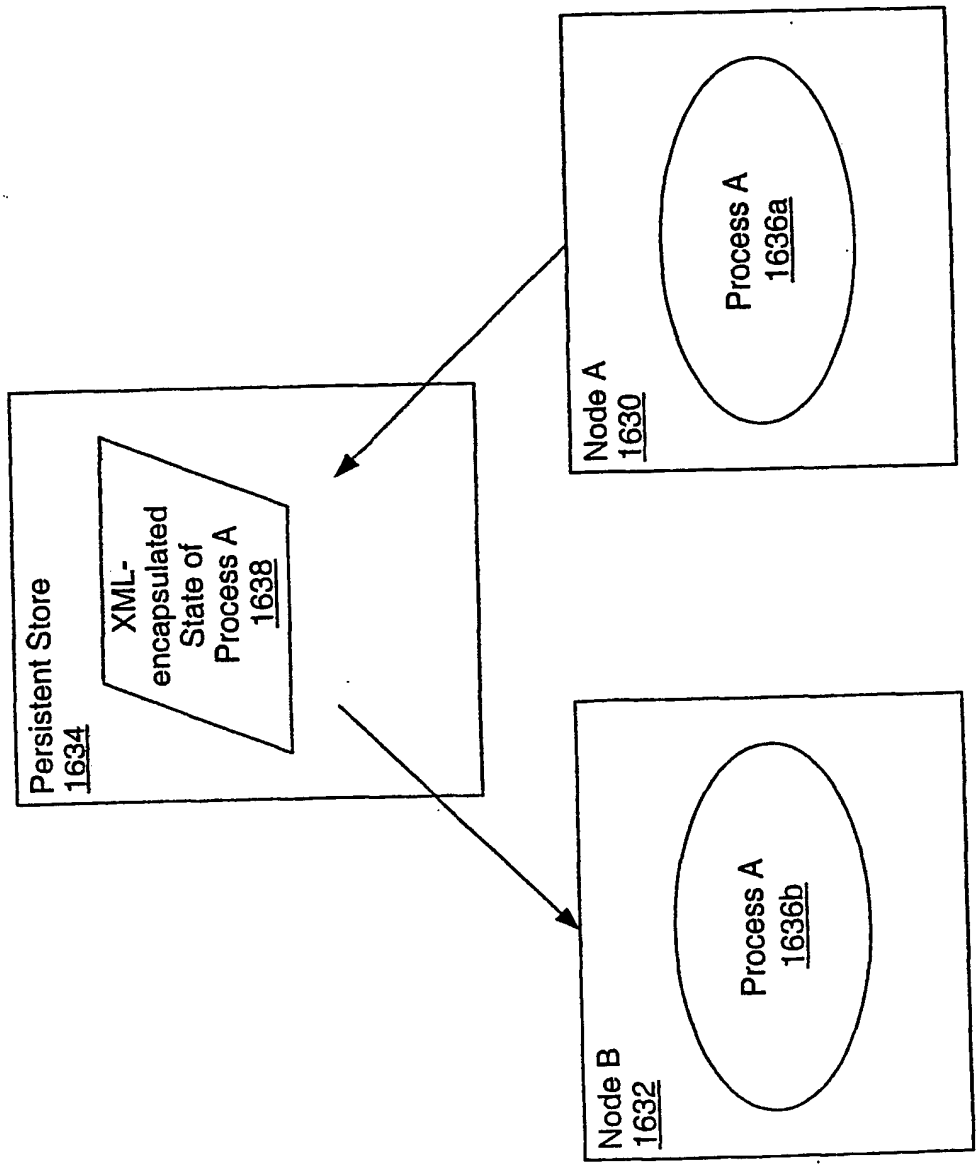


Fig. 37

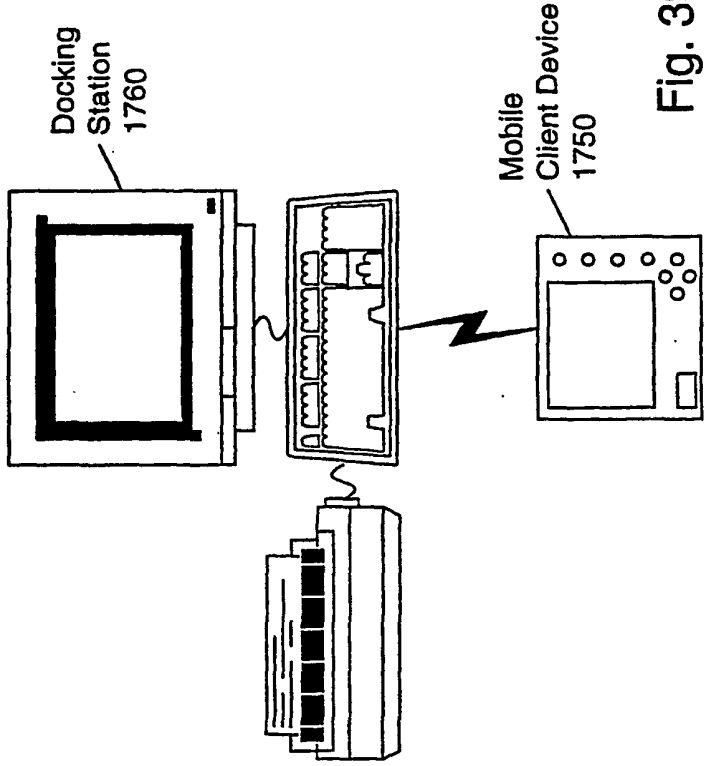


Fig. 39b

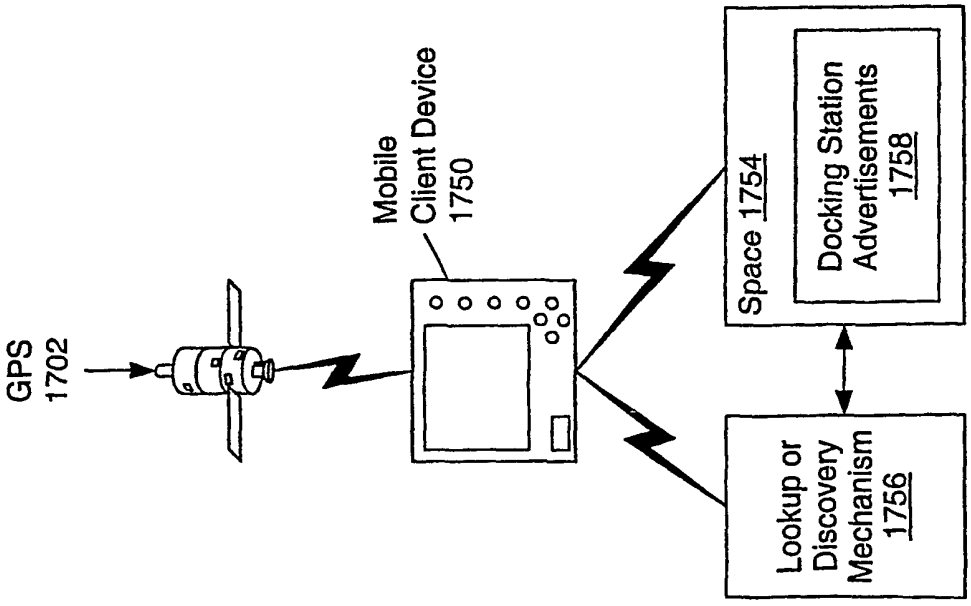


Fig. 39a

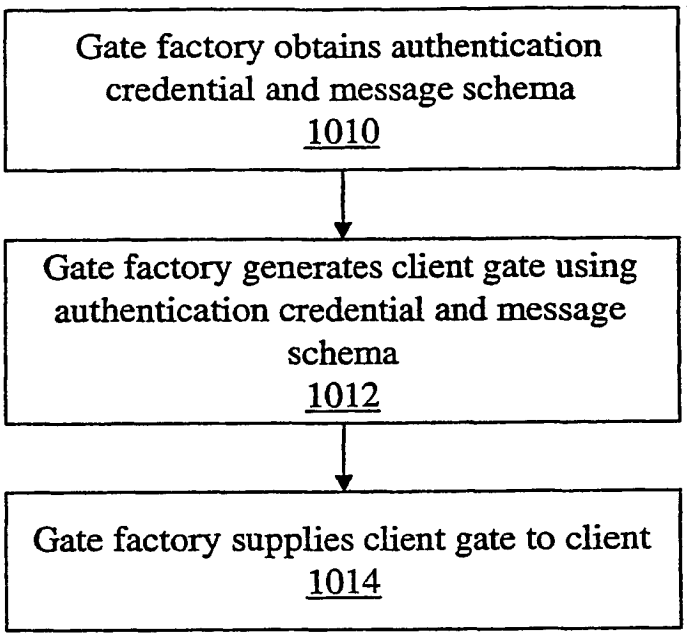


FIG. 41

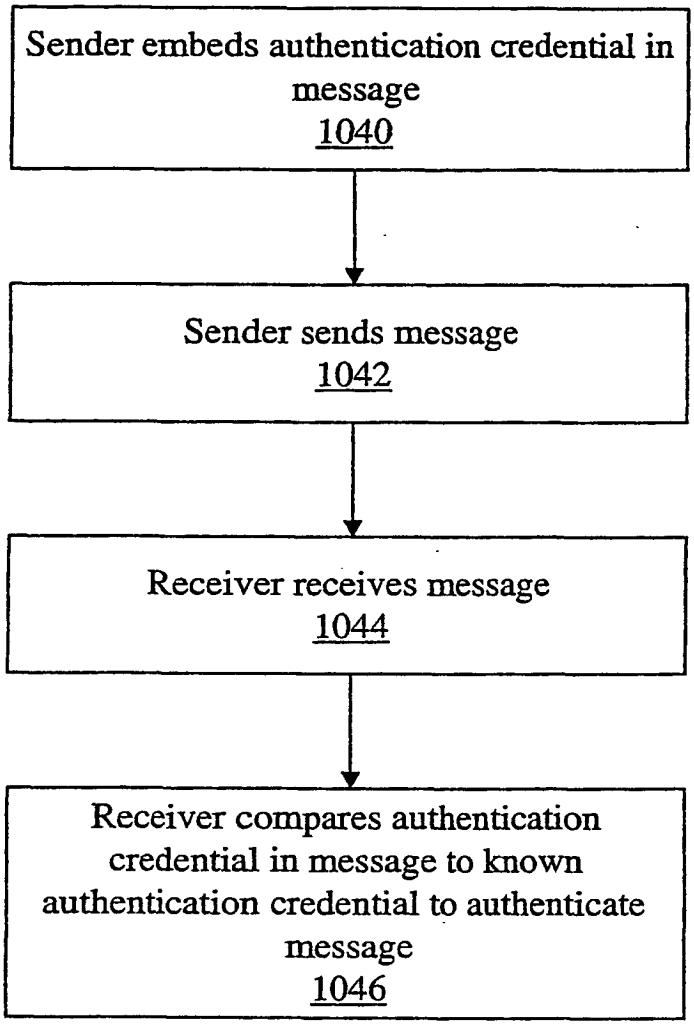


FIG. 42c

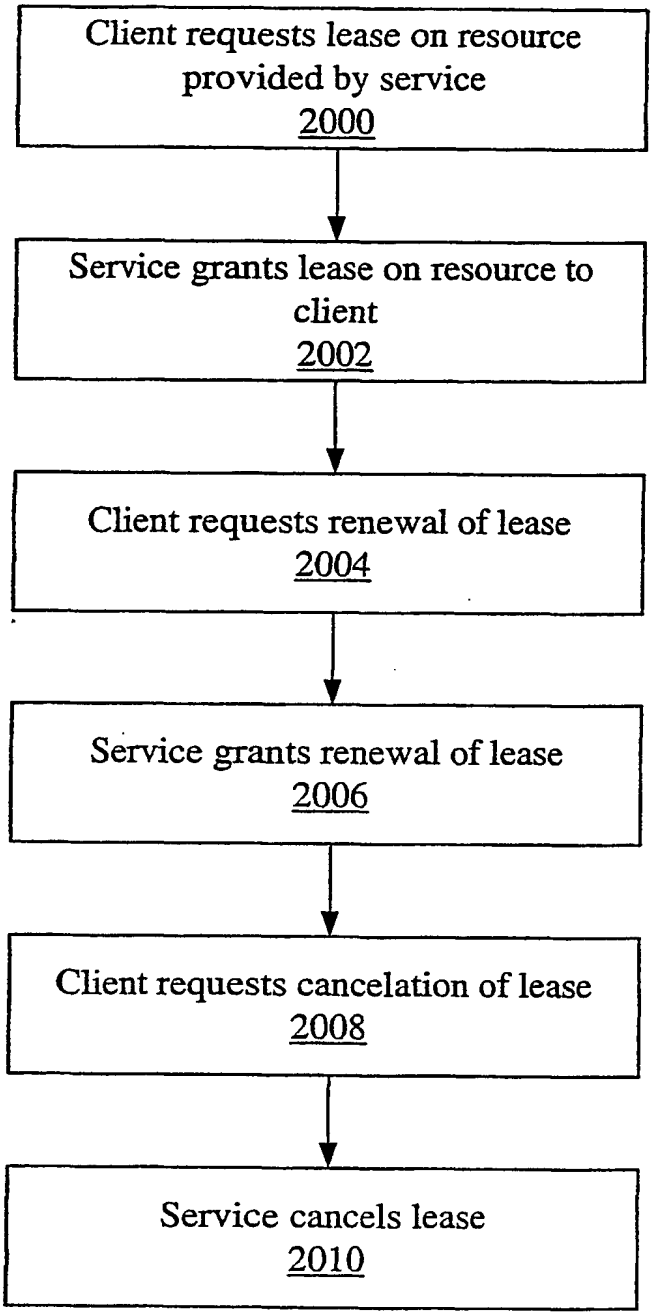


FIG. 44

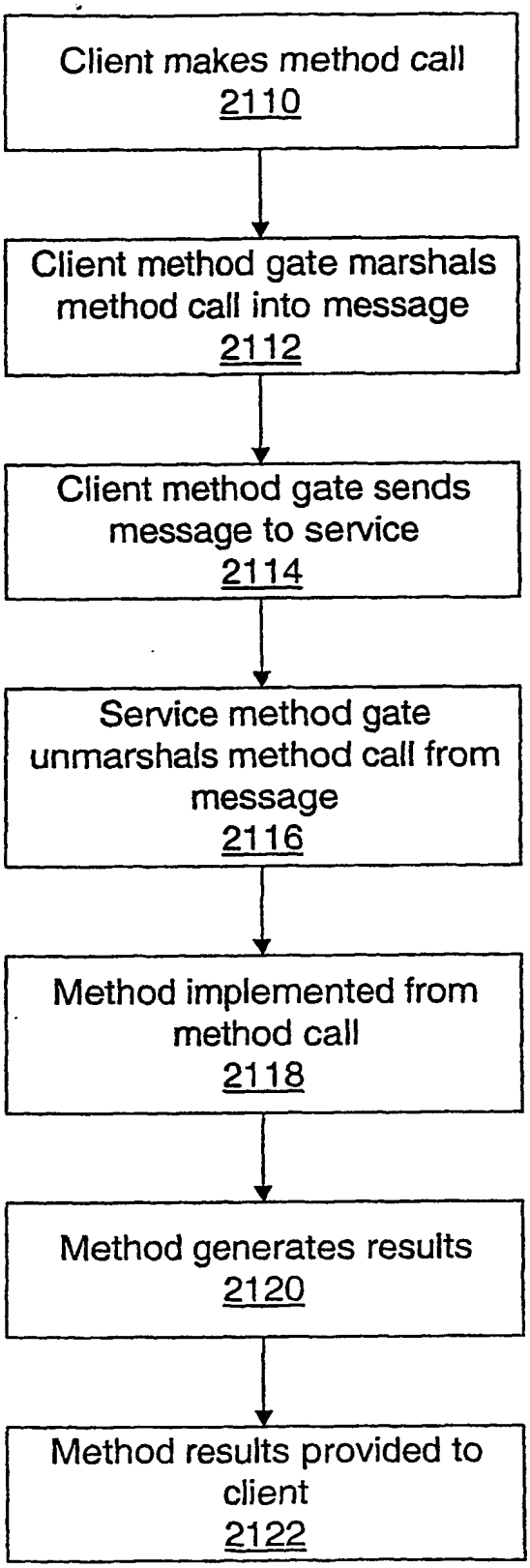


Figure 45b

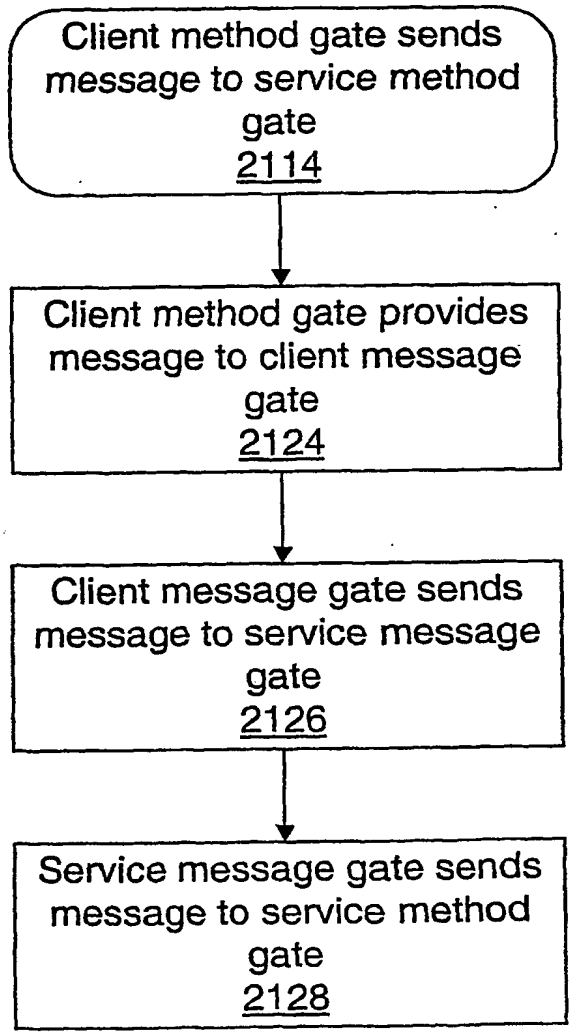


Figure 45c

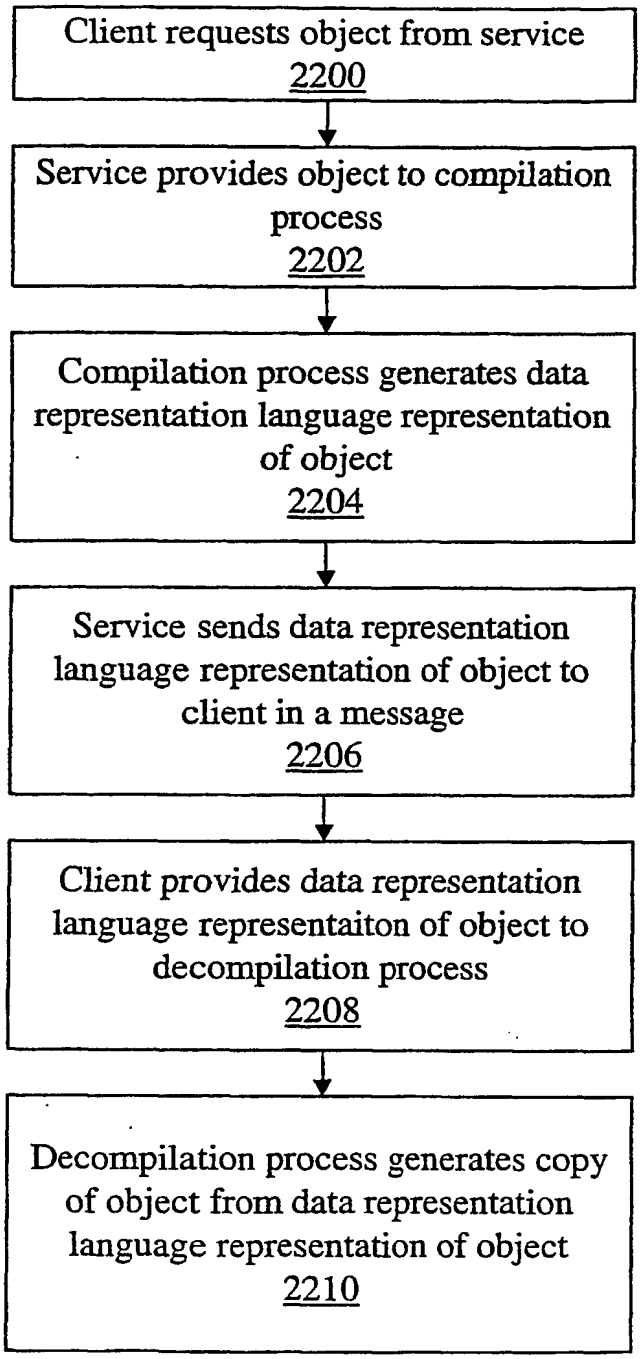


Figure 46

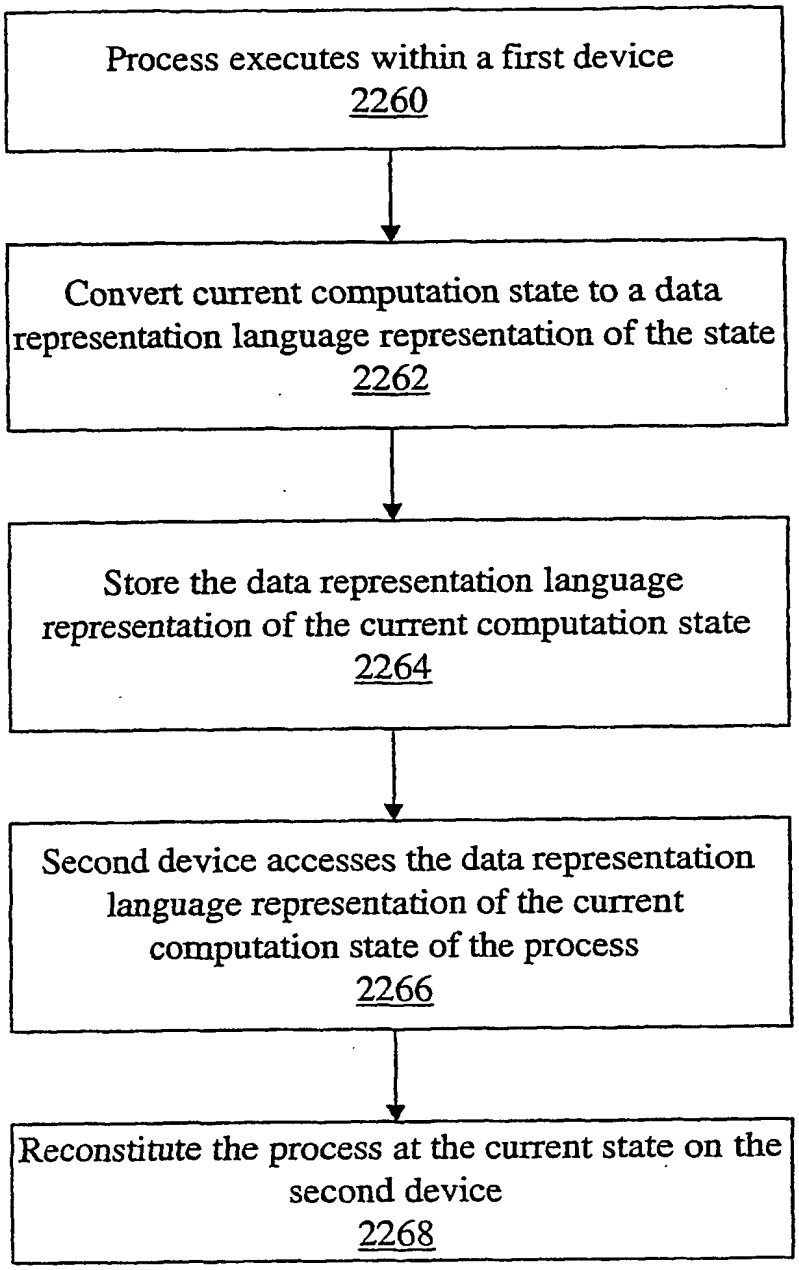


Figure 48

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
29 November 2001 (29.11.2001)

PCT

(10) International Publication Number
WO 01/090883 A3

(51) International Patent Classification[7]: G06F 9/46

(21) International Application Number: PCT/US01/15120

(22) International Filing Date: 9 May 2001 (09.05.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:

60/202,975	9 May 2000 (09.05.2000)	US
60/208,011	26 May 2000 (26.05.2000)	US
60/209,430	2 June 2000 (02.06.2000)	US
60/209,140	2 June 2000 (02.06.2000)	US
60/209,525	5 June 2000 (05.06.2000)	US
09/672,200	27 September 2000 (27.09.2000)	US

(71) Applicant: **SUN MICROSYSTEMS, INC.** [US/US]; 901 San Antonio Road, Palo Alto, CA 94303 (US).

(72) Inventors: **SLAUGHTER, Gregory, L.**; 3326 Emerson St., Palo Alto, CA 94306 (US). **SAULPAUGH, Thomas, E.**; 6938 Bret Harte Dr., San Jose, CA 95120 (US). **TRAVERSAT, Bernard, A.**; 2055 California St., Apartment 402, San Francisco, CA 94109 (US). **ABDELAZIZ, Mohamed, M.**; 78 Cabot Ave., Santa Clara, CA 95051 (US).

(74) Agent: **KOWERT, Robert, C.**; Conley, Rose & Tayon, P.C., P.O. Box 398, Austin, TX 78767-0398 (US).

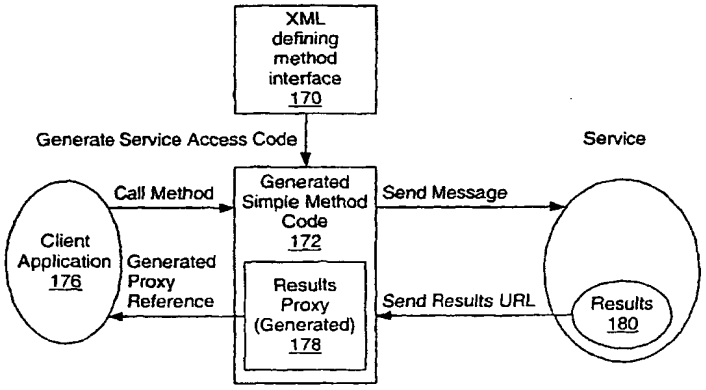
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *with international search report*

[Continued on next page]

(54) Title: REMOTE FUNCTION INVOCATION WITH MESSAGING IN A DISTRIBUTED COMPUTING ENVIRONMENT



(57) Abstract: An interface between clients and services in a distributed computing environment is described. Method gates may provide an interface to remotely invoke functions of a service. A method gate may be generated from an advertisement that may include definitions for one or more messages for remotely invoking functions of the service. A client may generate messages containing representations of method calls. The service may invoke functions that correspond to the set of messages. A method gate on the service may unmarshal the message and invoke the function. The client may receive the results of the function directly. Alternatively, the results may be stored, an advertisement to the results may be provided, and a gate may be generated to access the results. Message gates may perform the sending and receiving of the messages between the client and service. In one embodiment, functions of the service may be computer programming language (e.g. Java) methods. In one embodiment, a message including a representation of a method call may be generated when no actual method call was made. In one embodiment, a method call may be transformed into messages that may be sent to the service; the service may not know that the messages were generated from a method call. In one embodiment, a service may transform messages requesting functions into method calls; the client may not know that the service is invoking methods to perform the functions.

WO 01/090883 A3

INTERNATIONAL SEARCH REPORT

In ial Application No
PCT/US 01/15120

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 G06F9/46

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
EPO-Internal, IBM-TDB

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	WO 99 44121 A (SUN MICROSYSTEMS INC) 2 September 1999 (1999-09-02) page 7, line 23 -page 13, line 11; figure 3 --- -/--	1-4, 7-11, 16-20, 23-26, 32-34, 38-41, 45, 51-54, 56,57,60

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
15 October 2002	24/10/2002

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Michel, T

Form PCT/ISA/210 (second sheet) (July 1992)

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	MUELLER-WILKEN S ET AL: "XML and Jini - On Using XML and the JAVA Border Service Architecture to integrate mobile devices into the JAVA Intelligent Network Infrastructure" , 29 February 2000 (2000-02-29), XP002188507 the whole document	1,5,6, 17,21, 22,33, 35,36, 39,42, 43,46, 51,55
A	EDWARDS K: "Core jini, PASSAGE" CORE JINI, XX, XX, June 1999 (1999-06), pages 636-637,651-657, XP002212109 the whole document	12-14, 27,29, 30,44, 58,59